CS5800 Algorithms

Recitations from Oct 7, 2019 to Oct 11, 2019

Ravi Sundaram

Problem 1 (Huffman Coding)

Suppose we have some string message we would like to transmit. We can do something standard like ASCII, and have 7 bits represent each character. But depending on the message, we can economize the number of bits we have to use. Intuitively, in the vast majority of messages/languages, some letters will appear more often than others. In the English language, e appears a lot more often than z. So it makes sense if we used a few bits to represent e and a lot of bits to represent z. That's what Huffman coding does. And note, Huffman coding is a greedy algorithm.

1. Take the phrase 'learn northeastern'. Use Huffman coding to encode this. What is the total amount of bits used at the end? How many bits would we have used if we used 7 bits per character?

2. Say we use a min heap for our implementation. What is the runtime for this encoding procedure?

Solution:

1. We get 61 characters from the encoding, vs 126 if we didn't use anything.

2. $O(n \log n)$, where n is the number of distinct characters appearing in the string.

...

■

Problem 2 (Spanning Trees)

1. If a connected undirected graph has n vertices, show that any spanning tree for it has $n-1$ edges.

2. Prove that for any weighted undirected graph such that the weights are all distinct (no two edges have the same weight), the minimal spanning tree is unique.

3. Suppose we have a weighted undirected graph such that the weights are all distinct (no two edges have the same weight). And let e_{min} denote the edge with the minimum weight. Show that the minimum spanning tree will always contain e_{min}.

Solution:

1. To do this we use Problem 3 Part 1, and induction. We will prove that every tree on n vertices has $n-1$ edges. Base case with one vertex checks out. Suppose that any tree on k vertices has $k-1$ edges. Now suppose we take an arbitrary tree with $k+1$ vertices. Call this new graph T_{k+1} By Problem 3 Part 1, there is at least one vertex with degree 1. Let's call this vertex l, and the one edge e_l. Then if we delete e_l and l from T_{k+1}, we get a component with k vertices that's also a tree (since we started with a tree). By the inductive hypothesis,the number of edges is $k-1$ edges. Now the only way to include e is through e_l, so T_{k+1} has k edges. So we showed any tree on n vertices has $n-1$ edges. A spanning tree for a graph on n vertices will obviously be a tree on n vertices, so it must have $n-1$ edges as well.

2. Suppose the minimum spanning tree (MST) is not unique. Then there exists two distinct minimum spanning trees, that we call T_1, T_2. Let e_{min} denote the edge with smallest weight that's in exactly in one of the T_1, T_2 but not the other (such an edge must exist, otherwise T_1, T_2 have to be identical). Suppose without loss of generality its in T_1 (it could be in T_2 but then we use the same argument flipped). Then we take e_{min} and add it to T_2. Now note that T_2 contains n edges, so it must have a cycle (this is a good exercise). Let's call this cycle C. There must be an edge $e_2 \in C$, such that $e_2 \in T_2$ and $e_2 \notin T_1$ (if such an e_2 did not exist, then T_1 would contain a cycle which is impossible since T_1 is a tree). So e_2 is in a cycle that's exactly in one of T_1, T_2 but not the other. By definition of e_{min}, we know that $w(e_2) > w(e_{min})$. So if we remove e_2 from C, we get a new spanning tree (also a good exercise to check that this will indeed be a spanning ttree) whose weight is less than T_2. But this is impossible, since T_2 is a minimum spanning tree. Therefore, the minimum spanning tree must be unique.

3. Similar argument as last time. Suppose we have an MST T_1 that doesn't contain e_{min}. We can add e_{min} to T_1 and then remove any other edge from the cycle created, to get a new spanning tree with weight less than T_1. So we get a contradiction, since T_1 is supposed to be the minimum spanning tree, therefore e_{min} must be contained in the MST.

...

■

Problem 3 (Graphs)

1. Show that every tree has at least one vertex with degree 1.

2. A bipartite graph is a graph G whose vertices can be divided into two sets, S and T, so that the only edges in G are edges between one vertex in S and one vertex in T. Give an algorithm to check if a graph is bipartite.

3. Show that a graph is bipartite if and only if it doesn't have a cycle of odd length (a cycle with 3 edges is an example of a cycle of odd length).

Solution:

1. Choose any vertex from the tree. Then follow an arbitrary path, with the stipulation that you're not allowed to repeat edges. Since there are no cycles, once we've visited a vertex on our hypothetical path, we will never visit it again. Since there are a finite number of edges/vertices, our

path will eventually terminate. And the last vertex we land at will be a vertex of degree one. It must have degree at least one since we were able to reach it. But it can't have degree more than once since we have visited it for the first time and weren't able to choose any other edges.

2. You can run BFS, and at every level label all the vertices in the level either 1 or 2, and we alternate between 1 and 2 at every level. If we try to label the same vertex both 1 and 2, then graph isn't bipartite, otherwise it is.

3. First suppose the graph is bipartite. Let's say our two vertex sets are $V = (L \cup R)$. If we start on some vertex on the left side, to get back to a vertex on the left side we have to take two edges. So any cycle will be even.

   Now suppose every cycle of G is even. Let u be an arbitrary vertex. For any vertex v that's connected to u, let $d(u, v)$ denote the shortest distance between u and v (edges all have unit length). If $d(u, v)$ is even, give a label 2 to v. If the distance is odd, give a label 1 to v. Now note that if there are two vertices of the same label that have an edge between them, there must be a cycle of odd length since odd + odd = even + even = even. So then G is bipartite, and the labels denote the vertex sets.

...

■

Problem 4 (Extras...)

1. Suppose we have a weighted undirected graph G, where all the weights are not necessarily distinct. Give an efficient algorithm that, given an edge e, determines whether e is in a minimum spanning tree of G and whether e is in all minimum spanning trees of G.