

Due March 13, 2:45pm

Instructions: Same instructions as usual. Also, I will post detailed instructions on writing up your homework separately.

1. (20 pts.) Maximum spanning tree

In class we saw how to find a minimal spanning tree in a graph G . Suppose we now want to find a maximal spanning tree in G . The cost of a spanning tree is the same as always (it's the sum of the costs of the edges in the tree), but now we want to find a spanning tree whose cost is maximized. Describe an efficient algorithm to do this. You should be able to achieve a running time of $O(|E|\lg|V|)$.

2. (25 pts.) Maxi-min spanning tree

Comcast is building a cable network, which they want to use for broadcasting streaming video to n endpoints. We have a graph $G = (V, E)$, where each vertex $v \in V$ corresponds to an endpoint, and where each edge $e \in E$ has a (positive) capacity that represents the maximum data throughput that can be sent over that link. The capacity of a spanning tree is given by the capacity of the edge in the tree of lowest capacity, since that will be the bottleneck link. In other words,

$$\text{capacity}(T) = \min\{\text{capacity}(e) : e \in T\}.$$

Devise an efficient algorithm to find a maximal-capacity spanning tree in G . A running time of $O(|E|\lg|V|)$ should be achievable.

Hint 1: You may want to try some examples.

Hint 2: There are many approaches, but one possible approach is to modify a standard algorithm slightly. To prove your algorithm correct, you may need to modify the standard cut lemma appropriately.

3. (25 pts.) Clustering

To help recommend movies to its users that they might like, Netflix wants to find a way to partition its n users into k clusters. The basic idea is that each user will rate some of the movies he/she has seen; Netflix will construct the k clusters somehow, where hopefully all the users in a cluster have a similar taste in movies; and then Netflix will recommend movies to you based on ratings from other people in the same cluster as you. But to make this work, we need to figure out how to assign the Netflix users to clusters.

Suppose we are given some data on the users in terms of an undirected graph $G = (V, E)$ where each vertex $u \in V$ represents a Netflix user and each edge $(u, v) \in E$ has a length $\ell(u, v)$ that

represents the “dissimilarity” of users u and v (the larger $\ell(u, v)$, the less similar are their tastes). Also, suppose Netflix decides they would like to use the following criteria for selecting clusters. The *separation* of a particular assignment of users to clusters is given by

$$\text{separation} = \min\{\ell(u, v) : (u, v) \in E \text{ and users } u, v \text{ are not in the same cluster}\}.$$

Intuitively, if we’ve done a good job of choosing clusters, the clusters should be far apart, so the separation should be large. Come up with an assignment of the users into exactly k clusters, that maximizes the separation of this assignment.

Hint: Imagine running Kruskal’s algorithm, and think about the length of the $n - k + 1$ st edge added to its collection.

4. (30 pts.) Decoding

You work for NASA at the ground station associated with a interplanetary probe. The probe transmits scientific data back to Earth. If some bit of data gets corrupted in transmission, there’s no feasible way to request the probe to re-send that data, so to deal with this, the probe’s messages are encoded using an error-correcting code. Your job is to write an efficient decoder for this code.

More specifically, a n -bit message $m = (m_1, \dots, m_n)$ is encoded to a n -symbol codeword $c = (c_1, \dots, c_n)$ using the encoding process $e(\cdot)$, i.e., $c = e(m)$. Each symbol of the codeword is chosen from the alphabet $\{R, G, B, Y\}$. The encoding process encodes each bit of the message, bit-by-bit, in a stateful fashion: at each point it maintains a count of the number of 1-bits seen in the message so far, reduced modulo 4. More formally, we have state values s_0, s_1, \dots, s_n where $s_0 = 0$ and $s_i = s_{i-1} + m_i \pmod 4$. The i -th codeword symbol c_i is computed as a function of s_{i-1} and m_i : $c_i = f(s_{i-1}, m_i)$. The function f is as follows:

s_{i-1}	m_i	c_i
0	0	R
0	1	G
1	0	B
1	1	Y
2	0	G
2	1	R
3	0	Y
3	1	B

For instance, $e(011110) = RGYRBR$.

When the probe has a message m to send, it encodes it to get the codeword $c = e(m)$, then transmits c back to Earth. However, due to the long distances involved, there is a chance that some symbols of c might be received incorrectly. Let $\tilde{c} = (\tilde{c}_1, \dots, \tilde{c}_n)$ represent what is actually received by the ground station on Earth. Assume that each symbol of c has a 3% chance of being received incorrectly, and if it is received incorrectly, all 3 incorrect symbols are equally likely. In other words, if $c_i = R$, then with probability 0.97, $\tilde{c}_i = R$; with probability 0.01, $\tilde{c}_i = G$; with probability 0.01, $\tilde{c}_i = B$; and with probability 0.01, $\tilde{c}_i = Y$. Assume that all these probabilities are independent, so what happens to c_i is independent of what happens at all other positions of c .

Suppose you have received \tilde{c} . For any candidate message m , we could compute the probability of receiving \tilde{c} (what we actually got) assuming that the probe transmitted $e(m)$. Your job is to design an efficient algorithm to find the message m that maximizes this probability, given \tilde{c} .

One brute-force way to do this would be to enumerate over all 2^n possible messages m , compute the probability $\Pr[\tilde{c}|m]$ for each, and find the m that maximizes this probability. However, the brute-force method would take $\Theta(n \cdot 2^n)$ time, which is very inefficient. Your task is to come up with an $O(n)$ time algorithm. You can assume that you can perform basic mathematical operations (e.g., multiply, add, raise to a power, take a logarithm, etc.) on two numbers in $O(1)$ time.

Hint: Construct a dag $G = (V, E)$ that represents the set of all possible sequences of states that the encoder could go through. Your graph should be of size $O(n)$. Then, recall that you can compute shortest paths (or longest paths) in a dag, in $O(|V| + |E|)$ time. So, find a way to put lengths on each edge of your dag, so that this is just a shortest-paths (or longest-paths) problem.

P.S. Yes, this is actually used in practice for spacecraft-to-Earth communications, and in many other settings as well! Well, the specific code described above is not a very good code—one can do much better—but it is similar in spirit to what's used in practice, and the algorithms are the same. These same algorithms are also used by scientists: given a (noisy) sequence of observations of some physical phenomena, deduce the most likely explanation of what actually happened. So the algorithm you will devise here has widespread and wondrous applications.