CS 577: Introduction to Algorithms	10/12/2015

Homework 5 Solutions (Review Problems)

Instructor: Dieter van Melkebeek — TA: Kevin Kowalski

Problem 1

The claim seems reasonable enough, but one should also be skeptical about such claims. In this case, the claim is false. A counterexample is given in Figure 1.

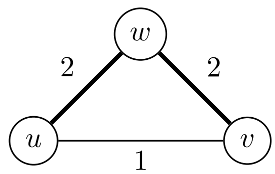

Figure 1: Counterexample in which T is the set of bold edges.

We explain how one can derive this counterexample. We need a graph that contains a cycle (since otherwise there is a unique minimum spanning tree), so we begin by considering the smallest cycle that is also a graph (i.e. we don't consider the cycle of length 1, which is a self loop, or a cycle of length 2, which are two parallel edges). As the question states, we need a graph in which the edge weights are not distinct (since otherwise there is a unique minimum spanning tree). But we cannot have all edges weights the same either (since otherwise every spanning tree is minimum), so we must have one unique edge weight x and two duplicate edge weights y. If $x > y$, then there is a unique minimum spanning tree, so we must have $x < y$.

At this point, we have only made necessary (minimal) choices in constructing a possible counterexample. Now one can check that these necessary (minimal) conditions are also sufficient. For concreteness, we pick $x = 1$ and $y = 2$. Any set of two edges in this graph defines a spanning tree. The two minimum spanning trees include the minimum weight edge (u, v). The spanning tree that is the counterexample to the claim is the set of bold edges with the duplicate weights.

Problem 2

The key observation is the following: Whenever there are three or more non-borrowed bottles, it never hurts to postpone borrowing empty bottles. We can formally use an exchange argument to justify this. Suppose we use $k \leq 3$ borrowed bottles to get another one. Because we have three or more non-borrowed bottles, we can replace these k borrowed bottles with non-borrowed ones and still get another one. In both cases, we get the same extra cola, so nothing is lost via this exchange.

Let $\text{OPT}(N)$ denote the maximum number of colas we can drink with N bought bottles. For $N \geq 3$, the above observation shows that

$$\text{OPT}(N) = 3 + \text{OPT}(N-2) \tag{1}$$

That is, drinking 3 bought bottles gives one back, and we can continue the process in the next round as if we had bought $N-2$ bottles, namely $N-3$ that were actually bought and the one that we got back in return for the 3 empty bottles. For the base cases, since $N \geq 1$, we need to consider $N = 1, 2$.

$N = 1$ We need to borrow at least two empty bottles in order to to get at least one extra bottle. However, if we borrow $k \geq 2$ empty bottles and use at least two of them to get an extra bottle, we have $(k+1) - 3 + 1 = k - 1$ bottles in total in the next round. Since our total number of bottles cannot increase over time, this means we'll be able to return at most $k - 1$ empty bottles at the end, whereas we need to return k. Thus, we cannot use any borrowed empty bottles, and the best we can do is just drink the one bottle we bought: $\mathrm{OPT}(1) = 1$.

$N = 2$ We can drink our two bottles, borrow an empty and turn in the three empty bottles for a new full one, drink the latter, and return it empty. This shows that $\mathrm{OPT}(2) \geq 3$. In order to do better, we'd need to turn in three empty bottles for a new full one at least twice, which would reduce our total number of bottles from $k + 1$ (k borrowed empty bottles and 2 full bought ones) to $k - 3$, which makes it impossible to return k empty bottles at the end. Thus, $\mathrm{OPT}(2) = 3$.

Solving (1) gives that

$$\mathrm{OPT}(N) = \begin{cases} 3n & \text{if } N = 2n \\ 3n+1 & \text{if } N = 2n+1 \end{cases}$$

for some $n \geq 1$. A single formula for this is $\lfloor \frac{3N}{2} \rfloor$ since

$$\left\lfloor \frac{3N}{2} \right\rfloor = \begin{cases} \left\lfloor \dfrac{3 \cdot 2n}{2} \right\rfloor = 3n & \text{if } N = 2n \\[2ex] \left\lfloor \dfrac{3 \cdot (2n+1)}{2} \right\rfloor = \left\lfloor 3n + \dfrac{3}{2} \right\rfloor = 3n+1 & \text{if } N = 2n+1. \end{cases}$$

Therefore given N, one can directly return $\lfloor \frac{3N}{2} \rfloor$.

Complexity Arithmetic operations can be done in time polynomial in the length of N (encoded in binary), therefore one can compute $\lfloor \frac{3N}{2} \rfloor$ in time polynomial in $\log N$, as desired.