Homework 20 sample solution

Due 11/30/16

November 22, 2016

Consider the following algorithm, which correctly solves the Bandersnatch (BS) problem using a solution to the JubJub (JJ) problem:

```
Input: data: array of positive integers
Input: n: size of data
Output: Bandersnatch(data)
Algorithm: BandersnatchReduction
Sort data
for i = 1 to n do
    if JubJub(data) then
        data[i] = data[n − i] − data[i]
    else
        data[i] = data[i] · data[n − i]
    end
    Sort data
end
return data
```

1. Suppose that BS is NP-Hard and JJ $\in P$. Prove that $P = NP$ <u>*or*</u> explain why BandersnatchReduction (BSR) does not prove $P = NP$.

   BSR *does* prove $P = NP$ in this case.

   Note that BSR takes $O(nJ(n))$ time, where $J(n)$ represents the worst-case complexity of JubJub on an array of size n. Since JJ $\in P$, $J(n)$ is polynomial, so $O(nJ(n))$ is also polynomial. Since there is a polynomial-time algorithm for an NP-Hard problem, every NP problem can be solved in polynomial time, so $P = NP$.

   Alternatively, note that BSR is a reduction from BS to JJ. Since BS is an NP-Hard problem, this implies that JJ must also be NP-Hard. Since JJ $\in P$, there must be a polynomial-time algorithm that solves JJ. In either case, we have a polynomial-time algorithm for an NP-Hard problem, so $P = NP$.

2. Suppose that BS $\in P$ and JJ is NP-Hard. Prove that $P = NP$ <u>*or*</u> explain why BSR does not prove $P = NP$.

BSR *does not* prove $P = NP$ in this case.

We know that BS can't take any longer than $O(nJ(n))$, where $J(n)$ represents the worst-case complexity of JubJub on an array of size n. However, $J(n)$ may or may not be polynomial, as we don't know whether JJ $\in P$ or not, and since BS $\in P$, there almost certainly a more efficient algorithm for BS that doesn't involve an NP-Hard algorithm.

On the flip side, we know that JJ can't finish any sooner than $\Omega(b(n)/n)$, where $b(n)$ represents the lower bound for the worst-case complexity of Bandersnatch on an array of size n. Since BS $\in P$, $b(n)$ must be polynomial, but we can't tell whether JJ is polynomial or not; it just takes longer than the polynomial $b(n)/n$.