# EDA HW1

B02901178 江誠敏

October 13, 2015

Prob.1 It is obvious that the algorithm has to run in $\Omega(n)$ time, since each vertex would be visit once. We only remain to proof that the algorithm runs in $O(n)$ time, and so runs in $\theta(n)$ time.

First we proof that we would walk down each edge only one time. Notice that we would walk down only when preforming Tree-Minimum. Let $e = (v_0, v_1)$ be an edge and that $v_1$ is the parent of $v_0$. consider the vertex sequence $v_0, v_1, \cdots, v_n$, where $v_i$ is the parent of $v_{i-1}$, and $n$ is the smallest integer such that $v_{n-1}$ is the right child of $v_n$ (i.e., for every $i < n$, $v_{i-1}$ is the left child of $v_i$) . We claim that we will walk down the edge $e$ only in case #1 when preforming Tree-Successor. This is because it is the only condition we would call Tree-Minimum. And when we are preforming case #1, we would first go down to the right child, and then we repeatly go down to the left child. So if we would reach $v_0$ after such operations start from $u$, $u$ must be the first "right" parent of $v_0$, and this $u$ is unique, namely $v_n$. Hence each edge could be walked down only once.
Now, we could show that each edge could be walked upward only once too. Since every path from $v_1$ to $v_0$ on tree have to pass through $e$, we could conclude that each time we walk upward from $v_0$ to $v_1$, we would have to first walk down $v_1$ to $v_0$ before, so an edge could be walked upward at most once.

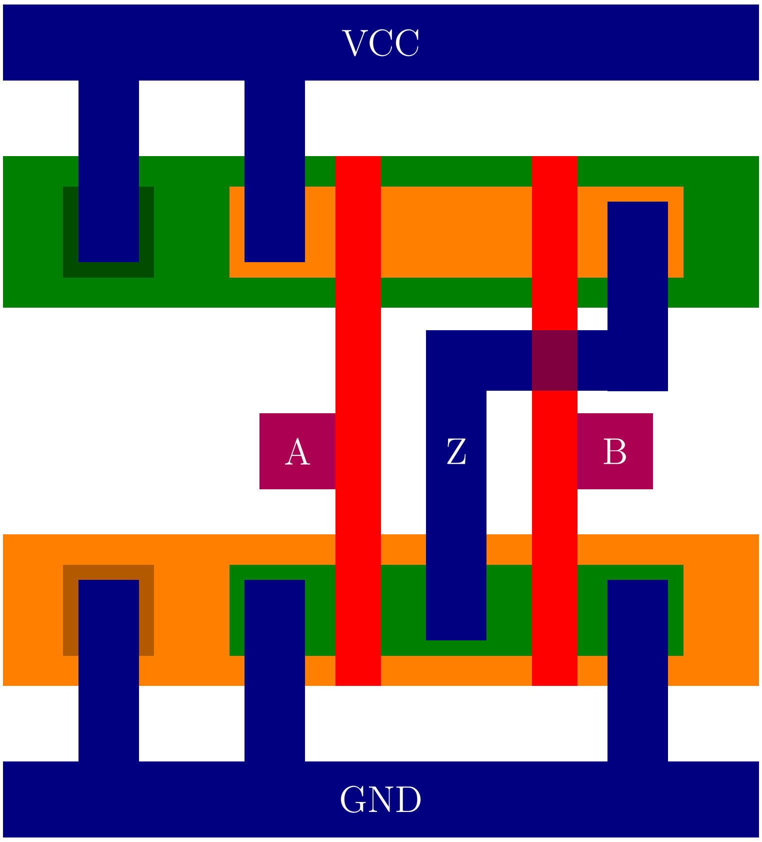
So each edge would be walked at most twice, and since $|E| = |V| - 1 = O(n)$ in tree, we conclude that the operations the algorithm take is $2 \cdot O(n) = O(n)$, and hence the proof is complete.

Prob.2 Let $T(n)$ be the time needed if we partition array in 3 parts. Then, when merging 3 sorted sub-arrays, we would have to find the minimum element among the top element of 3 arrays, which would take 2 comparison. And since we have $n$ elements totally, the comparison in merging is $2n = O(n)$. so the recursive formula is

$$T(n) = 3T(n/3) + O(n)$$

Solve the recursion and we get $T(n) = O(n \log_3(n)) = O(n \log_2(n))$, since $\log_3(n) = (\log(2)/\log(3)) \log_2(n)$. So the time complexity is the same as if we partition the array in 2 parts. It is not faster.

Prob.3 The answer is the figure below.




Prob.4 The answer is the figure below.