CS 371 HW 5

Josh Blaz

3/20/19

1. Chapter 6, Exercise 1.

   (a) An example where this algorithm wouldn't work is we had a Graph G where: $v_1 = 8$, $v_2 = 10$, and $v_3 = 6$. This algorithm would choose v_2, and remove the neighbors from G, but the actual maximum weight independent set would be v_1 and v_3, as it would add up to 14, which is greater than 10, which is the weight the algorithm would come up with.

   (b) Use graph G with the following weights: $v_1 = 5$, $v_2 = 15$, $v_3 = 6$, $v_4 = 1$, $v_5 = 6$, $v_6 = 1$, and $v_7 = 5$. In this case, the algorithm would choose the odd numbered vertices ($v_1 + v_3 + v_5 + v_7 = 24$) over the even numbered vertices ($v_2 + v_4 + v_6 = 17$). However, this is wrong, because the independent set with the maximum weight is v_2, v_5, and v_7 which adds up to a weight of 28.

   (c) OPT(i) = The weight of the maximum weight independent set from nodes 1 to i.
   The recursive definition is as follows: OPT(i) = 0 if $i = 0$, OPT(i) = w_1 if $i = 1$, and OPT(i) = max(OPT[$i-2$]+w_i, OPT[$i-1$]) otherwise. Following this recursive definition, the final solution can be found at OPT(n), once we've iterated through and computed the OPT value at every node.
   Because OPT computes the final solution at OPT(n), this algorithm will run at O(n) time.

2. Chapter 6, Exercise 2.

   (a) This algorithm doesn't account for the case in which there's a high-stress job in week 1 (no preparation required) that has a payout greater than the payout of a low-stress job in the same week.

   (b) OPT(i) = The value of the highest value plan of jobs 1 to i.
   The recursive definition is as follows: OPT(i) = 0 if $i = 0$, OPT(i) = max(l_1,h_1) if $i = 1$, and OPT(i) = max(l_i + OPT($i-1$), h_i + OPT($i-2$)) otherwise. Following this recursive definition, the final solution can be found at OPT(n), once we've iterated through and computed the OPT value for every job being considered.

Given that the OPT computes the final solutions at OPT(n), when every job is considered, this algorithm will run at $O(n)$.

3. Chapter 6, Exercise 4.

   (a) This algorithm would not work in the following situation: where $n = 3$, $M = 10$, NY = [4,1,10], and SF = [1,4,3]. This wouldn't work, because the company would choose the following order: SF, NY, SF. This would give an operating cost of 5, however, the moving costs would be 20, so in total this algorithm would choose a plan costing 25. The optimal solution would be to work in San Francisco for all 3 months, and only spend 8, with no moving costs.

   (b) An example where every optimal plan would move three times is: where $n = 4$, $M = 10$, NY = [10,1,10,1] and SF = [1,10,1,10]. The optimal solution would be to use the following plan: SF, NY, SF, NY. This plan adds up to a cost of 34 (1 + 1 + 1 + 1 + 3 switches). This property exists, because in this case, where $M = 10$, it is more worth it to move and incur the cost of 10, because there's the opportunity to only have a cost of 1 for every move.

   (c) $OPT_{NY}(i)$ = The value of the minimum cost for the first i months spent operating in NY.
   $OPT_{SF}(i)$ = The value of the minimum cost for the first i months spent operating in SF.
   The recursive definition is as follows: $OPT(i) = 0$, if $i = 0$, $OPT(i) = OPT_{NY}[i] + \min(OPT_{NY}[i-1], OPT_{SF}[i-1]+M)$, if ends in "NY", and $OPT(i) = OPT_{SF}[i] + \min(OPT_{SF}[i-1], OPT_{NY}[i-1] + M)$ if ends in "SF".
   Following this recursive definition, the final solution is found by taking the minimum of $OPT_{NY}(n)$ and $OPT_{SF}(n)$, which are computed once all months have been iterated through at which each OPT value is computed. Given that an algorithm using this OPT definition would reach completion once $OPT_{SF}(n)$ and $OPT_{NY}(n)$ are reached, the algorithm would complete in $O(n)$ time.

4. Code.