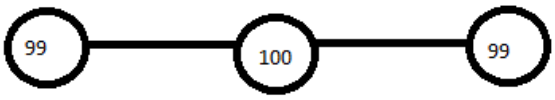
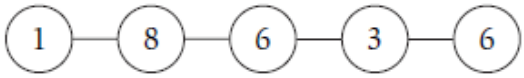
David Sun
CMPS 102
Homework 5

Problem 1: KT, Ch6, #1, p. 315
Counter-Example for part A: Consider the graph below.



If we pick the maximum weight node and delete its neighbors, our subset of nodes is the node containing 100 only, while the maximal subset is the two nodes with weight 99.

Counter-Example for part B: Consider the graph below.



Taking the subset containing odd numbered nodes gives us a subset with weight 13, and taking the subset with even number nodes gives us a subset with weight 11. However, the maximum weight subset is 8 and 6 with weight 14.

What are the subproblems and how are they related
In this problem, the subproblems we want to solve consider the maximum value for the graph ending at a particular index.
Every subproblem considers the value of the maximum subset of the previous subproblem. If i is the current node index, then the maximum subset from node 1 to node i either takes node i or takes the optimal sum from node 1 to node $i-1$, depending on which subset has maximal sums. Each subproblem is related in this case because at each step we are considering the maximum weight of the current value with the previous subproblems.

Recurrence
The optimal sum for a set of nodes from 1 to i can be given by the recurrence

$$OPT(i) = \begin{cases} 0 & i = 0 \\ max(v_i + OPT(i-2), OPT(i-1)) & otherwise \end{cases}$$

If we're at node i, we can either add its weight to $OPT(i-2)$ or continue using $OPT(i-1)$ because we cannot use two nodes directly adjacent to each other. Note that this recurrence does not return the subset of the actual nodes but rather the maximal sum of the nodes. We can use it, however, to find the actual subset.

Tables
Our table will be an array consisting of of the max value of the subsets. Index i of the table contains the max value sum for node 1 to node i. The table A will be built bottom up and each entry in the table is equal to $\max(vi + A[i-2],\ A[i-1])$, where v_i is the weight of node i.

Algorithm and Time Bound

At this point, we've only computed the optimal sum for a given index i, we haven't actually found the subset of nodes. To find the actual subset of nodes, we can use the following algorithm: Start with an empty set s for the set of nodes. Lookup the value of the last index in our table A, call this index i. While $i > 1$, check if $A[i]$ is equal to $A[i-1]$. If such is the case, the $i-1$th node was taken rather than the ith node, so append node $i-1$ to s and set $i := i-3$. Otherwise, the ith node was taken so append node i to s and set $i := i-2$. At the end of the loop if $i = 1$, the first node was taken, so append node 1 to s.

Filling out the table has an $O(n)$ runtime. Notice that the recurrence depends on $OPT(i-2)$ and $OPT(i-1)$. Without any sort of memoization, a calculation for $OPT(i)$ would take $O(2^n)$ time. However, the table A keeps track of $OPT(i-2)$ and $OPT(i-1)$ in the cells directly previous, which can be looked up in $O(1)$ time. This reduces the recurrence to $O(n)$ running time. The algorithm to check which nodes belong or do not belong to the maximum subset in the best case skips over 3 nodes each time in the best case and skips over 2 nodes each time in the worst case. In this case, the algorithm runs in $n/2$ time worst case and $n/3$ time best case. Both of these times are still linear so the entire algorithm from filling the table and appending to the set takes total of $n + n/2$ or $n + n/3$, both of which are $O(n)$.

Problem 2: KT, Ch6, #4, p. 315

(a) Counter example for given algorithm

SF = [1, 2]
NY = [2, 1]
Moving cost = 3.
The algorithm given in the book would choose SF in Month 1 and NY in Month 2, with the cost being $1 + 1 + 3 = 5$. The optimal plan would be just to stay in SF, with the cost being $1 + 2 = 3$.

(b) Example in which every optimal plan must change locations at least 3 times

Consider the following costs for NY and SF

```
NY = [1, 11, 1, 11]
SF = [11, 1, 11, 1]
Moving cost = 9
```

In month 1 to month 2, you move from NY to SF. In month 2 to month 3, you move from SF to NY. In month 3 to month 4, you move from NY to SF, thus there are three moves. The moves are optimal because the sum of the moving cost with the operating cost at a given location is still cheaper than having stayed at that location for the previous months. In our example, this is true for every move.

What are the subproblems and how are they related

Each subproblem for month i is the minimum operating cost of the business for a given location at month i. These subproblems are related because each cost at month i is the minimum cost of operating in that location at that month. This means that the following month must consider whether continuing operation in the same location is most optimal, or would it have been more optimal to have changed locations.

Recurrence

Since there are two locations to consider, we need two recurrences, one for SF and one for NY. The optimal cost for SF at month i can be given by the recurrence:

$$SF(i) = \begin{cases} SF[1] & i = 1 \\ v_i + min(SF[i-1], NY[i-1] + C_m) & otherwise \end{cases}$$

Where C_m is the cost of moving and v_i is the cost of operating in SF at month i only. Likewise, the optimal cost for NY at month i can be given by the recurrence:

$$NY(i) = \begin{cases} NY[1] & i = 1 \\ v_i + min(NY[i-1], SF[i-1] + C_m) & otherwise \end{cases}$$

Where C_m is the cost of moving and v_i is the cost of operating in NY at month i only. Thus for a give month i, the optimal cost can be given by $\min(SF[i], NY[i])$

Tables

Since we have two recurrences, we need two arrays to represent the optimal cost of operating in each city at month i. The table will be initialized in a bottom-up manner. The first entry in the two tables contain the cost of operating in each city for the first month. Following entries in the table for SF follows the recurrence for SF, and following entries in the table for NY follows the recurrence for NY.

Algorithm and Time Bound

We will start with two arrays NY and SF both of size n where n is the number of months. Let C be the cost of moving. The first entry in the SF array contains the cost of operating in SF for month 1 and the first entry in the NY array contains the cost for operating in NY for month 1. Let SF_i and NY_i be the cost for operating in SF and NY in month i alone respectively. Then from $i = 2$ until $i = n$,
$SF[i] = SF_i + \min(SF[i-1], NY[i-1] + C)$ and
$NY[i] = NY_i + \min(NY[i-1], SF[i-1] + C)$.
At the end of the algorithm, to find the optimal cost, take $min(SF[n], NY[n])$.

For each index i, we make one calculation for $SF[i]$ and $NY[i]$. These values at i depend on $NY[i-1]$ and $SF[i-1]$. These values are memoized so they can be found in $O(1)$ time. Thus, for a total of n steps and 2 calculations taking constant time at each step, our algorithm runs in $2n$ time, and $2n = O(n)$.

Problem 3:

(a) The maximum sum for the contiguous subsequence in the array is 55.

(b) What are the subproblems and how are they related

The subproblems in this case consider the sum of the current contiguous sequence along with the current global max sum of any previous subsequence. They are related in the sense that if the sum of the current sequence is greater than the previous max, the global max is replaced with the sum of the current sequence. Otherwise, the global max stays the same.

Recurrence

This problem will have two recurrences, one to keep track of the current positive sum of the subsequence ending at an index i, and one to keep track of the global maximum value ending at index i.