Name: Daniel Kim
ID: 102353420

CSCI 3104, Algorithms **Profs. Chen & Grochow**
Problem Set 5 – Due Thurs Feb 20 11:55pm **Spring 2020, CU-Boulder**

Advice 1: For every problem in this class, you must justify your answer: show how you arrived at it and why it is correct. If there are assumptions you need to make along the way, state those clearly.

Advice 2: Informal reasoning is typically insufficient for full credit. Instead, write a logical argument, in the style of a mathematical proof.

Instructions for submitting your solutions:

- All submissions must be easily legible.
- You should submit your work through the **class Canvas page** only.
- You may not need a full page for your solutions; pagebreaks are there to help Gradescope automatically find where each problem is. Even if you do not attempt every problem, please allot at least as many pages per problem (or subproblem) as are allotted in this template.

Quicklinks: 1a 1b 2a 2b 3a 3b 3c

---

1. *In this question we consider the change-making problem (as covered in recitation), of making change for n cents using the smallest number of coins. Suppose we have coins with denominations of $v_1 > v_2 > \cdots > v_r$ for r coins types, where each coin's value v_i is a positive integer. Your goal is to obtain a set of counts $\{d_i\}$, one for each coin type, such that $\sum_{i=1}^{r} d_i = k$ and where k is minimized, and such that the sum of the values $\sum_{i=1}^{r} d_i v_i = n$.*

   (a) *A greedy algorithm for making change is the **cashier's algorithm**. Consider the following pseudocode—meant to implement the cashier's algorithm—where n is the amount of money to make change for and v is a vector of the coin denominations:*

```
change(n,v,r) :
   d[1 .. r] = 1       // initial histogram of coin types in solution
   while n > 0 {
      k = r
      while ( k > 0 and v[k] > n ) { k++ }
      if k==0 { return 'no solution' }
      else { n = n - v[k] }
   }
   return d
```

Name: Daniel Kim
ID: 102353420

This code has bugs. Identify the bugs and explain why each would cause the algorithm to fail.

- First of all, there is a bug in d[1...r] = 1 because initial histogram of coin types should start at 0, not 1. If it does start at 1, it means that each types of the coins will be counted as 1 already before making the change.
- Next, I would put k = r and the if statement (if k == 0) and before the outer while loop because if it's in the inner while loop, k will never be 0 so it will never execute and k = r will keep repeating also.
- It should be k−−, not k++ because k starts at the largest value which is r so every iteration it needs to subtracted
- Lastly, I would get rid of the else statement and add n = n - v[k] into the inner while loop

Name: Daniel Kim
ID: 102353420

(b) *Identify a set of Euro coin denominations (a subset of the denominations here)*[1] *for which the greedy algorithm does not yield an optimal solution for making change. Justify your answer in terms of optimal substructure and the greedy-choice property. (The set should include the 1 Euro cent so that there is a solution for every value of n.) Include an example where the cashier's algorithm with your choice of denominations yields a set of coins that is larger than it needs to be, and also show the smaller set of coins adding up to the same value.*

Set of Euro coin denominations: 1, 2, 5, 10, 20, 50, $c1$, $c2$

Example:
Total money: 80
Using subset of Euro coin: 1, 2, 20, 50
Cashier's Algorithm: $80 = 50 + 20 + 2 + 2 + 2 + 2 + 2$ (7 coins)
Optimal solution: $80 = 20 + 20 + 20 + 20$ (4 coins)

Optimal substructure property and greedy choice property needs to hold true in order for the greedy algorithm yield an optimal solution for making change. Greedy choice property can be constructed incrementally (and a partial solution assigned a score) without reference to future decisions, past decisions, or the set of possible solutions to the problem. There is always an optimal solution that starts with greedy choice. In cashier's algorithm, it chooses the largest coins first and will continue until it reaches the smallest coins, which is the greedy choice property. However, from the example from above, greedy choice property doesn't always end up with optimal solution. In order words, this means that the greedy choice property cannot apply to this algorithm. Because the greedy choice property does not work, this algorithm does not always yield an optimal solution.

[1] `https://www.google.com/search?q=euro+coin+denominations`

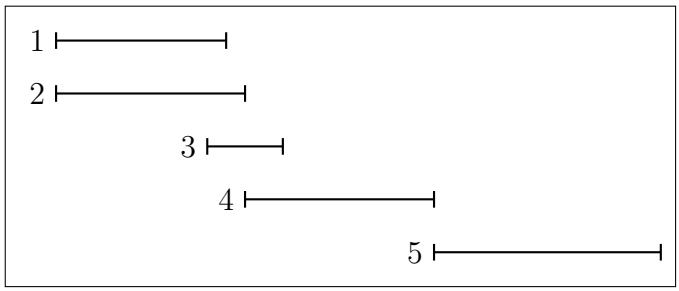
Name: Daniel Kim
ID: 102353420

2. *In this question we consider the interval scheduling problem, as covered in class.*

   (a) *Consider a greedy algorithm which always selects the shortest appointment first. Draw an example with at least 5 appointments where this algorithm fails. List the order in which the algorithm selects the intervals, and also write down a larger subset of non-overlapping intervals than the subset output by the greedy algorithm.*

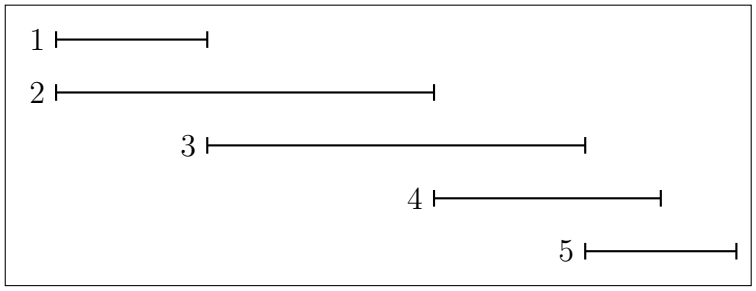
   ***A comment on level of justification (applies to all of problems 2 and 3):** to help us understand your thinking, it is worth writing a little about the order in which the algorithm selects the intervals. For example "The algorithm takes intervals in the order [1,3,4]: first the algorithm takes interval 1 because that is the shortest. Interval 1 conflicts with intervals 2 and 5, so they are removed. The next shortest is 3, which conflicts with interval 6, and the only remaining interval is 4."*

   YOUR ANSWER HERE. We've provided you with some sample code to draw such a figure natively in LaTeX, you just need to modify the start/end points and possibly add more intervals if you need (as well as answer the rest of the question!).



   Order: The first algorithm takes interval 3 because it is the shortest. Interval 3 conflicts with intervals 4 and 2 and 1, so they are skipped and the next shortest is 5. The algorithm takes intervals in the order [3 5]

(b) *Consider a greedy algorithm which always selects the longest appointment first. Draw an example with at least 5 appointments where this algorithm fails. Show the order in which the algorithm selects the intervals, and also show a larger subset of non-overlapping intervals than the subset output by the greedy algorithm. The same comments apply here as for 2a in terms of level of explanation.*

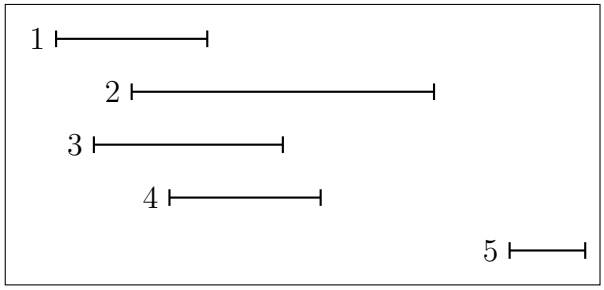


If the algorithm chooses the longest interval first, it will only take interval 2 first and then end with interval 4, and the total time cost is 9. However, if we take the shorter intervals, interval 1 will be first, interval 3 and ends with interval 5 which the total time cost will end with 10.

Name: Daniel Kim
ID: 102353420

3. *In this question we'll consider weighted problems.*

   (a) *Consider the* weighted *interval scheduling problem. In this problem, the input is a list of n intervals-with-weights, each of which is specified by* $(start_i, end_i, wt_i)$*. The goal is now to find a subset of the given intervals in which no two overlap and to maximize* the sum of the weights*, rather than the total number of intervals in your subset. That is, if your list has length n, the goal is to find* $S \subseteq \{1, \ldots, n\}$ *such that for any* $i, j \in S$*, interval i and interval j do not overlap, and maximizing* $\sum_{i \in S} wt_i$*. Consider the greedy algorithm for interval scheduling from class, which selects the job with the earliest end time first. Give an example of weighted interval scheduling with at least 5 intervals where this greedy algorithm fails. Show the order in which the algorithm selects the intervals, and also show a higher-weight subset of non-overlapping intervals than the subset output by the greedy algorithm. Same comments apply as on problem 2.*



Assuming:
Interval 1: weight 1
Interval 2: weight 10
Interval 3: weight 1
Interval 4: weight 1
Interval 5: weight 1

Order: The algorithm takes interval 1 first since it ends the earliest. Interval 2, interval 3, and interval 4 overlaps with interval 1 which means that these conflicts with interval 1. Because there are conflicts, the next shortest is 5. Therefore, the algorithm takes intervals in the order: [1 5]

Analysis: Because interval 2 weights more than the sum of the rest the intervals and skips, the example above is not a optimal solution.

A higher-weight subset: [2 5].

(b) *Consider the Knapsack problem: the input is a list of n items, each with a value and weight* (val_i, wt_i)*, and a threshold weight* W*. All values and weights are strictly positive. The goal is to select a subset* S *of the items such that* $\sum_{i \in S} wt_i \leq W$ *and maximizing* $\sum_{i \in S} val_i$*. (Note that,* unlike *change-making, here there is only one of each item, whereas in change-making you in principle have an unlimited number of each type of coin.) Consider a greedy algorithm for this problem which makes greedy choices as follows: among the remaining items, choose the item of maximum value such that it will not make the total weight exceed the threshold* W*. Give an example of knapsack with at least 5 items where this greedy algorithm fails. Show the order in which the algorithm selects the items, and also show a higher-value subset whose weight does not exceed the threshold. Same comments apply as on problem 2.*

Assuming threshold weight is 20, and:
Item 1: weight 20, value 20
Item 2: weight 1, value 10
Item 3: weight 1, value 10
Item 4: weight 1, value 10
Item 5: weight 1, value 10

Order: The algorithm takes item 1 first because of its maximum value and the total weight exceed the threshold. After picking item 1 the algorithm will stop because there are no room for any more weight. Therefore, the algorithm takes item in the order: [1]

Analysis: The value to weight ratio is only 1 even though item 1 has the maximum value. The value to weight ratio will get higher if we pick any other items. Therefore, this solution is not optimal.

A higher-weight subset: [2 3 4 5]

(c) *Now consider the following algorithm for the knapsack problem. Call the* relative value *of item i the ratio val_i/wt_i. Consider the greedy algorithm which, among the remaining items, chooses the item of maximum* relative value *such that it will not make the total weight exceed the threshold W. Give an example of knapsack where this greedy algorithm fails. Show the order in which the algorithm selects the items, and also show a higher-value subset whose weight does not exceed the threshold. Same comments apply as on problem 2.*

Assuming threshold weight is 20, and:
Item 1: weight 1, value 1, relative value 1.
Item 2: weight 1, value 2, relative value 2.
Item 3: weight 1, value 3, relative value 3.
Item 4: weight 1, value 20, relative value 20.
Item 5: weight 20, value 200, relative value 10.

Order: The algorithm takes item 4 first because item 4 does not make the total weight exceed the threshold due to item 4 is the max relative value has the most relative value. Next, the maximum relative value is item 5 after item 4 however, we need to skip item 5 because the current capacity is 19 and the weight for item 5 is 20. After that, it will keep picking the next biggest item which item 3 to item 2 to item 1. The algorithm takes items in the order: [4 3 2 1]

Analysis: This solution is not optimal. Item 4 has the maximum relative value, and after taking item 4 out, we don't have more for item 5 which is the next maximum total value. Also in this case, the items are not divisible.

A higher-weight subset: [5]

Name: Daniel Kim
ID: 102353420

CSCI 3104, Algorithms **Profs. Chen & Grochow**
Problem Set 5 – Due Thurs Feb 20 11:55pm **Spring 2020, CU-Boulder**

COLLABORATED WITH:
RUIJIANG MA
JINGQI LIU
HASSAN ALSAHLI