ADA Tutorial 2 - Tuesday

August 12, 2025

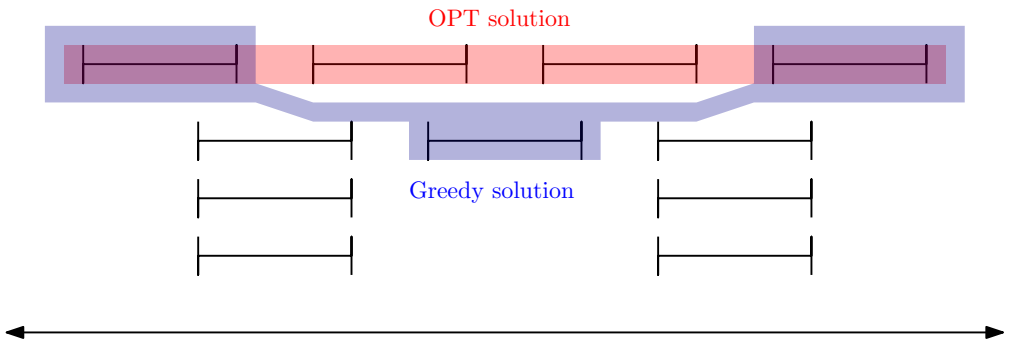
Question 1

Consider the following problem:
INPUT: A set $S = (x_i, y_i), 1 \leq i \leq n$ of intervals over the real line.
OUTPUT: A maximum cardinality subset S_0 of S such that no pair of intervals in S_0 overlap.
Consider the following algorithm:

- **Repeat until S is empty**
- **Select the interval I that overlaps the least number of other intervals.**
- **Add I to the final solution set S_0.**
- **Remove all intervals from S that overlap with I.**

Prove or disprove that this algorithm solves the problem.

Proof. See counterexample: □



Question 2

Consider the following Interval Coloring Problem.
INPUT: A set $S = (x_i, y_i), 1 \leq i \leq n$ of intervals over the real line. Think of interval (x_i, y_i) as a request for a room for a class that meets from time x_i to time y_i.
OUTPUT: Find an assignment of classes to rooms that uses the fewest number of rooms.
Note that every room request must be honored and that no two classes can use a room at the same time. Consider the following iterative algorithm. Assign as many classes as possible to the first room (we can do this using the greedy algorithm discussed in class), then assign as many classes as possible to the second room, then assign as many classes as possible to the third room, etc. Does this algorithm solve the Interval Coloring Problem? Justify your answer.

Proof. See counter-example: □

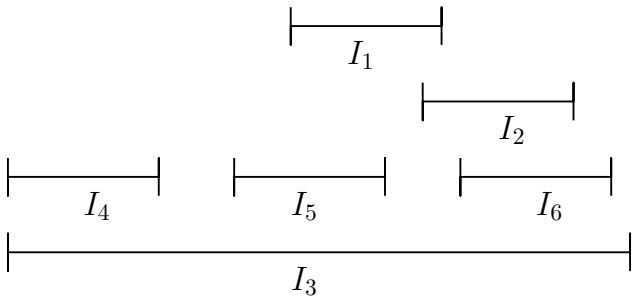


Figure 1: Assignment that uses fewest number of classes $\{I_1, I_4, I_6\}, \{I_2, I_5\}, \{I_3\}$. However, our greedy algorithm returns $\{I_4, I_5, I_6\}, \{I_1\}, \{I_2\}, \{I_3\}$, which is not optimal. Read about greedy graph coloring via maximum independent set extraction.

Question 3

a)The input to this problem is an integer L. The output should be the minimum cardinality collection of coins required to make L shillings of change (that is, you want to use as few coins as possible). The coins are worth $1, 5, 10, 20, 25, 50$ Shillings. Assume that you have an unlimited number of coins of each type. Formally prove or disprove that the greedy algorithm (that takes as many coins as possible from the highest denominations) correctly solves the Change Problem. So for example, to make a change for 234 Shillings the greedy algorithms would take four 50 shilling coins, one 25 shilling coin, one 5 shilling coin, and four 1 shilling coins.

Proof. Consider the case where $L = 40$. The greedy agorithm takes as many as possible from largest denomination to smallest.

- Take 0 of 50 as $50 > 40$.

- Take 1 of 25, remaining $40 - 25 = 15$.
- Take 0 of 20 as $20 > 15$.
- Take 1 of ten, remaining $15 - 10 = 5$.
- Take 1 of 5, remaining 0.

So the greedy algorithm produces the multi-set $\{25, 10, 5\}$.
But there is a better solution of $20 + 20 = 40$. □

b) The input to this problem is an integer L. The output should be the minimum cardinality collection of coins required to make L nibbles of change (that is, you want to use as few coins as possible). Now the coins are worth $1, 2, 2^2, 2^3, \ldots 2^{1000}$ nibbles. Assume that you have an unlimited number of coins of each type. Prove or disprove that the greedy algorithm (that takes as many coins of the highest value as possible) solves the change problem.
HINT: The greedy algorithm is correct for one of the above two subproblems and is incorrect for the other. For the problem where greedy is correct, use the following proof strategy: Assume to reach a contradiction that there is an input I on which greedy is not correct. Let $OPT(I)$ be a solution for input I that is better than the greedy output $G(I)$. Show that the existence of such an optimal solution $OPT(I)$ that is different than greedy is a contradiction. So what you can conclude from this is that for every input, the output of the greedy algorithm is the unique optimal/correct solution

Proof. The greedy algorithm would begin by taking $\lfloor \frac{L}{2^{1000}} \rfloor = P$ number of coins of 2^{1000} denomination.
Let $T = L \mod 2^{1000}$. Suppose we represent the greedy and optimal solutions by 1000 dimensional vectors $\vec{g}$ and $\vec{o}$ such that the i^{th} entry of $\vec{g}$ and $\vec{o}$ are the number of coins of denomination 2^{1000-i} in the greedy and optimum solution respectively.
Note that $g[0] = P$ and for $i \geq 1$, $g[i] \leq 1$.

Claim 1. *For $i \geq 1$, $o[i] \leq 1$.*

Proof. Assume that for some j, $o[j] \geq 2$. Then, by decreasing $o[j]$ by 2 and increasing $o[j-1]$ by 1, we get the same sum but reduce the number of coins by 1. Hence this solution is not optimal. □

Let i be the first index at which the vectors $\vec{g}$ and $\vec{o}$ differ. Note that the sum of all coins of denomination less than 2^{1000-i} in both the greedy and opt solutions is less that 2^{1000-i}.
Thus if $g[i] > o[i]$, then the value of the optimum solution is less than the greedy solution which is L.
Similarly, if $g[i] < o[i]$, then the value of the optimal is higher which is a contradiction.
Therefore, $\vec{g}$ $\vec{o}$ are the same and greedy is optimal. □

Question 4

You wish to drive from point A to point B along a highway minimizing the time that you are stopped for gas. You are told beforehand the capacity C of your gas tank in liters, your rate F of fuel consumption in liters/kilometer, the rate r in liters/minute at which you can fill your tank at a gas station, and the locations $A = x_1, x_2, \ldots, x_n = B$ of the gas stations along the highway. So, if you stop to fill your tank from 2 liters to 8 liters, you would have to stop for $6/r$ minutes. Consider the following two algorithms:
a. Stop at every gas station and fill the tank with just enough gas to make it to the next gas station.
b. Stop if and only if you don't have enough gas to make it to the next gas station, and if you stop,fill the tank up all the way.
For each algorithm either prove or disprove that this algorithm correctly solves the problem. Your proof of correctness must use an exchange argument.

Proof. The time required to fill gas is proportional to the amount of gas filled. Hence minimizing the time is the same as minimizing amount of gas used.
Strategy [b] fills the tank fully every time we stop for gas. If C= 50L and it takes 20L to go from A to C and 40 L to go from B to C, then we would fill the tank at B and have 10L remaining when we reach C. This strategy is not optimum as we could have filled up lesser amount at B.
Let g_i(resp. $o[i]$) be the gas filled at the i^{th} station in strategy[a](resp. lexicographically optimal solution). Let i be the first index at which g and o differ. *Note that the gas tank is empty when the car reaches the i^{th} station in both solutions.*

1. $g_i > o_i$: Since g_i is the smallest amount of gas needed to reach the next station, if o_i is less than g_i, optimum is not feasible.

2. $g_i < o_i$ In this case we can reduce o_i by 1 and increase o_{i+1} by 1 to get a lexicographically smaller optimal solution. Doing so does not make the solution infeasible since we are taking exactly the amount of gas to reach the next station.

Since we arrive at a contradiction in both cases, the greedy solution is identical to the optimum solution. □

Question 5

Consider the following problem. The input is a collection $A = \{a_1, \ldots a_n\}$of n points on the real line. The problem is to find a minimum cardinality collection S of unit intervals that covers every point in A. Another way to think about this same problem is the following. You know a collection of times (A) that trains will arrive at a station. When a train arrives there must be someone manning the station. Due to union rules, each employee can work at most one hour at the station. The problem is to find a schedule of employees that covers all the times in A and uses the fewest number of employees.

(a) Prove or disprove that the following algorithm correctly solves this problem. Let I be the interval that covers the most number of points in A. Add I to the solution set S. Then recursively continue on the points in A not covered by I.
(b) Prove or disprove that the following algorithm correctly solves this problem. Let $a_j$be the smallest (leftmost) point in A. Add the interval$I = (a_j, a_j + 1)$to the solution set S. Then recursively continue on the points in A not covered by I.
HINT: One of the above greedy algorithms is correct and one is incorrect for the other. The proof of correctness must use an exchange argument.

Proof. (a) Consider $A = \{0.5, 1, 1.25, 1.75, 2, 2.5\}$. The algorithm proposed would first pick the interval $[1, 2]$ and two more intervals to cover 0.5 and 2.5. But two intervals $[0.5, 1.5]$ and $[1.75, 2.75]$ would suffice.
(b) Let $g_1, g_2 \dots g_k$ be the start times of the intervals chosen by the greedy algorithm and $o_1, o_2 \dots o_m$ be the start times of the intervals chosen by the optimal algorithm. Assume both are in increasing order and let i be the first index in which they differ.

1 $g_i < o_i$**:** From our greedy algorithm it follows that there is a point at g_i which is not covered by the interval starting at $o_{i-1} = g_{i-1}$. This means the opt solution does not cover this point which makes it infeasible.

2 $g_i > o_i$**:** From greedy algorithm it follows that there is no point in the interval $[o_{i-1} + 1, g_i]$. Hence o_i can be increased to g_i without making it infeasible.

Therefore this greedy strategy is optimal. □

Question 6

We consider a greedy algorithm for two related problems
a.) The input to this problem consists of an ordered list of n words. The length of the i^{th} word is w_i. That is, the i^{th} word takes up w_i spaces. (For simplicity assume that there are no spaces between words.) The goal is to break this ordered list of words into lines; this is called a layout. Note that you cannot reorder the words. The length of a line is the sum of the lengths of the words on that line. The ideal line length is L. No line may be longer than L, although it may be shorter. The penalty for having a line of length K is $L - K$. The total penalty is the sum of the line penalties. The problem is to find a layout that minimizes the total penalty. Prove or disprove that the following greedy algorithm correctly solves this problem.

- **For i= 1 to n**
- **Place the i^{th} word on the current line if it fits.**
- **else place the i^{th} word on a new line;**

Proof. Minimizing the penalty is the same as minimizing the number of lines. Let g_i (resp. o_i) be the number of words assigned to the line i by the greedy algorithm (resp. optimal strategy). We consider that the optimal solution lexicographically maximizes the vector $(o_1, o_2 \dots o_p)$. Let i be the smallest index at which $g_i \neq o_i$.

1 If $o_i > g_i$ then the greedy algorithm would have added the extra words to the line i which is a contradiction.

2 If $g_i > o_i$, then by moving a word from line $i+1$ to line i, the optimal solution could have obtained a lexicographically maximal solution, which is a contradiction.

Therefore, the vectors g and o are identical, making greedy optimal.

□

b)The input to this problem consists of an ordered list of n words. The length of the i^{th} word is w_i. That is, the i^{th} word takes up w_i spaces. (For simplicity assume that there are no spaces between words.) The goal is to break this ordered list of words into lines; this is called a layout. Note that you cannot reorder the words. The length of a line is the sum of the lengths of the words on that line. The ideal line length is L. No line may be longer than L, although it may be shorter. The penalty for having a line of length K is $L-K$. The total penalty is the maximum of the line penalties. The problem is to find a layout that minimizes the total penalty. Prove or disprove that the following greedy algorithm correctly solves this problem.

- **For i= 1 to** n
- **Place the** i^{th} **word on the current line if it fits.**
- **else place the** i^{th} **word on a new line;**

Proof. Suppose $L = 9$ and the word lengths are $\{2, 2, 2, 2, 5\}$. The greedy algorithm would arreange $l_1 = \{2, 2, 2, 2\}$ and $l_2 = \{5\}$, which would incur a penalty of 4, while the optimum penalty is 3. □

Question 7

Consider the following problem. The input consists of the lengths $l_1, \dots, l_n$, and access probabilities$p_1, \dots, p_n$, for n files $F_1, \dots, F_n$. The problem is to order these files on a tape so as to minimize the expected access time. If the files are placed in the order $F_{s_1}, \dots F_{s_n}$then the expected access time is $\sum_{i=1}^{n} p_{s_i} \sum_{j=1}^{i} l_{s_j}$. Note that the term $p_{s_i} \sum_{j=1}^{i} l_{s_j}$ is the probability that you access the i^{th} file times the length of the first i files. For each of the algorithms below, either give a proof that the algorithm is correct using an exchange argument, or a proof that the algorithm is incorrect.

a. Order the files from shortest to longest on the tape. That is, $l_i < l_j$ implies that $s_i < s_j$.

b. Order the files from most likely to be accessed to least likely to be accessed. That is, $p_i < p_j$ implies $s_i > s_j$.

c. Order the files from the smallest ratio of the length over access probability to the largest ratio of the length over access probability. That is,$\frac{l_i}{p_i} < \frac{l_j}{p_j}$implies that $s_i < s_j$.

Proof. a. Counterexample: For $n = 2$ let $l_1 = 1$, $p_1 = 0$ and $l_2 = 2$, $p_2 = 1$. algorithm orders it as 1,2 with expected access time $0 \times 1 + 1 \times 3 = 3$. But the ordering 2,1 has expected access time $1 \times 2 + 0 \times 3 = 2$.

b. Counterexample: For $n = 2$ consider $l_1 = 1, p_1 = \frac{2}{3}$ and $l_2 = 0, p_2 = \frac{1}{3}$. Algorithm orders it as 1,2 with expected access time $\frac{2}{3} \times 1 + \frac{1}{3} \times 1 = 1$. But the ordering 2,1 has expected access time $\frac{2}{3}$.

c. Consider an optimal ordering of the files and suppose there exists as pair such that $\frac{l_i}{p_i} > \frac{l_{i+1}}{p_{i+1}}$ where file $i+1$ immediately follows file i in the optimal ordering.
The expected access time for these two files is $p_i(l_i + L) + p_{i+1}(l_i + l_{i+1} + L)$, where $L = \sum_{j=1}^{i-1} l_j$.
If we were to swap the files in the ordering, the expected access time is $p_{+1}(l_{i+1} + L) + p_i(l_i + l_{i+1} + L)$.
Note that the expected access for all other files is unchanged. The difference between the access times

$$p_i(l_i + L) + p_{i+1}(l_i + l_{i+1} + L) - p_{+1}(l_{i+1} + L) + p_i(l_i + l_{i+1} + L) = [\frac{l_{i+1}}{p_{i+1}} - \frac{l_i}{p_i}]p_i.p_{i+1}$$

which is negative. This implies the swap reduces the expected accesss time which implies $\forall i$, $\frac{l_i}{p_i} < \frac{l_{i+1}}{p_{i+1}}$. This in turn implies that in the optimal ordering files are arranged by increasing $\frac{l}{p}$ ratio.

□