

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' of S such that no pair of intervals in S' overlap.

Consider the following algorithm:

Repeat until S is empty

1. Select the interval I that overlaps the least number of other intervals.
2. Add I to final solution set S' .
3. Remove all intervals from S that overlap with I .

Prove or disprove that this algorithm solves the problem.

† † †

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 being 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.

- (a) 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, and in the class notes), 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.
- (b) Consider the following algorithm. Process the classes in increasing order of start times. Assume that you are processing class C . If there is a room R such that R has been assigned to an earlier class, and C can be assigned to R without overlapping previously assigned classes, then assign C to R . Otherwise, put C in a new room. Does this algorithm solve the Interval Coloring Problem? Justify your answer.

HINT: Let s be the maximum number of intervals that overlap at one particular point in time. Obviously, you need at least s rooms. Therefore any algorithm that uses only s rooms is obviously optimal. This lower bound on the number of rooms required allows you to prove optimality without using an exchange argument.

††

3. Consider the Change Problem in Austria. 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). In Austria 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 change for 234 Shillings the greedy algorithm would take four 50 shilling coins, one 25 shilling coin, one 5 shilling coin, and four 1 shilling coins.

†

4. Consider the Change Problem in Binaryland. 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). In Binaryland the coins are worth $1, 2, 2^2, 2^3, \dots, 2^{1000}$ 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 in Binaryland.

HINT: The greedy algorithm is correct for one of the above two problems 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.

††

5. 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 $A = x_1, \dots, B = x_n$ 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.

6. Consider the following problem. The input is a collection $A = \{a_1, \dots, a_n\}$ of n points on the real line. The problem is to find a minimum cardinality collection S of unit intervals that cover 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 scheduling 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 can be done using an exchange argument.

††

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

$$\sum_{i=1}^n p_{s(i)} \sum_{j=1}^{s(i)} \ell_{s(j)}$$

Don't let this formula throw you. The term $p_{s(i)} \sum_{j=1}^{s(i)} \ell_{s(j)}$ is the probability that you access the i th file times the length of the first i files.

For each of the below algorithms, either give a proof that the algorithm is correct, or a proof that the algorithm is incorrect.

- (a) Order the files from shortest to longest on the tape. That is, $\ell_i < \ell_j$ 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 that $s(i) > s(j)$.
- (c) Order the files from smallest ratio of length over access probability to largest ratio of length over access probability. That is, $\frac{\ell_i}{p_i} < \frac{\ell_j}{p_j}$ implies that $s(i) < s(j)$.

††

8. Consider the following problem. The input consists of n skiers with heights p_1, \dots, p_n , and n skies with heights s_1, \dots, s_n . The problem is to assign each skier a ski to minimize the average difference between the height of a skier and his/her assigned ski. That is, if the i th skier is given the $\alpha(i)$ th ski, then you want to minimize:

$$\frac{1}{n} \sum_{i=1}^n |p_i - s_{\alpha(i)}|$$

- (a) Consider the following greedy algorithm. Find the skier and ski whose height difference is minimized. Assign this skier this ski. Repeat the process until every skier has a ski. Prove or disprove that this algorithm is correct.
- (b) Consider the following greedy algorithm. Give the shortest skier the shortest ski, give the second shortest skier the second shortest ski, give the third shortest skier the third shortest ski, etc. Prove or disprove that this algorithm is correct.

HINT: One of the above greedy algorithms is correct and one is incorrect for the other. The proof of correctness can be done using an exchange argument.

††

9. 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 can not 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 ith word on the current line if it fits
    else place the ith word on a new line

```

††

10. 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 can not 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 ith word on the current line if it fits
    else place the ith word on a new line

```

HINT: The greedy algorithm is correct for one of the above two problems and is incorrect for the other. The proof of correctness can be done using an exchange argument.

††

11. We consider the following problem:

INPUT: A collection of jobs J_1, \dots, J_n , where the i th job is a 3-tuple (r_i, x_i, d_i) of non-negative integers.

OUTPUT: 1 if there is a preemptive feasible schedule for these jobs on one processor, and 0 otherwise. A schedule is *feasible* if every job J_i is run for x_i time units between its release time r_i and its deadline d_i .

We consider greedy algorithms for solving this problem that schedule times in an online fashion, that is the algorithms are of the following form:

```

t = 0;
while there are jobs left not completely scheduled
    pick a job  $J_m$  to schedule at time t;
    increment t;

```

One can get different greedy algorithms depending on how job J_m is selected. For each of the following method of selecting J_m , prove or disprove that the resulting greedy algorithms produce feasible schedules, if they exist for the jobs being considered.

- (a) Among those jobs J_i such that $r_i \leq t$, and that have not been scheduled for enough time units, pick J_m to be the job with minimal deadline. Ties may be broken arbitrarily. We call this algorithm EDF.
- (b) Among those jobs such that $r_i \leq t$, and that have not been scheduled for enough time units, pick J_m to be the job that has minimal remaining processing time. Ties may be broken arbitrarily. If a job have been executed for y time units before time t , then its *remaining processing time* is $x_i - y$. We call this algorithm SRPT.

As an example of EDF and SRPT consider the following instance: $J_1 = (0, 100, 1000)$, $J_2 = (10, 10, 100)$, $J_3 = (10, 10, 101)$, and $J_4 = (115, 10, 130)$.

EDF runs J_1 from time 0 through time 10. From time 10 until time 20, EDF runs J_2 because J_2 's deadline, 100, is less than J_1 's deadline, 1000, and less than J_3 's deadline, 101. From time 20 until time 30, EDF runs J_3 because J_3 's deadline, 101, is less than J_1 's deadline, 1000. From time 30 to time 115, EDF runs J_1 . From time 115 to time 125, EDF runs J_4 since J_4 's deadline, 130, is less than J_1 's deadline, 1000. EDF then runs J_1 from time 125 to time 130. Thus for this input, EDF finishes all the jobs by their deadline.

SRPT runs J_1 from time 0 through time 10. From time 10 until time 30, SRPT runs J_2 and J_3 (either can go first) because throughout this time period, J_2 and J_3 's remaining processing times are always equal and less than J_1 's remaining processing time, 90. From time 30 to time 115, SRPT runs J_1 . From time 115 to time 120, SRPT runs J_1 since J_1 's remaining processing time throughout this period is smaller than J_4 's remaining processing time, 10. From time 120 to 130, SRPT runs J_4 . Thus for this input, SRPT finishes all the jobs by their deadline.

12. We consider the following problem:

INPUT: A collection of jobs J_1, \dots, J_n , where the i th job is a 2-tuple (r_i, x_i) of non-negative integers.

OUTPUT: A preemptive schedule S that minimizes $\sum_{i=1}^n C_i$, where C_i is the first time that job J_i has been run for x_i time units after time r_i .

Give a greedy algorithm for this problem and prove that it is correct. See the previous problems for background.

13. Consider the following problem.

INPUT: Positive integers r_1, \dots, r_n and c_1, \dots, c_n .

OUTPUT: An n by n matrix A with 0/1 entries such that for all i the sum of the i th row in A is r_i and the sum of the i th column in A is c_i , if such a matrix exists.

Think of the problem this way. You want to put pawns on an n by n chessboard so that the i th row has r_i pawns and the i th column has c_i pawns.

Consider the following greedy algorithm that constructs A row by row. Assume that the first $i - 1$ rows have been constructed. Let a_j be the number of 1's in the j th column in the first $i - 1$ rows. Now the r_i columns with maximum $c_j - a_j$ are assigned 1's in row i , and the rest of the columns are assigned 0's. That is, the columns that still needs the most 1's are given 1's. Formally prove that this algorithm is correct.

HINT: Use an exchange argument.

† † †

14. Consider the generalization of the U2 bridge crossing problem where n people with speeds s_1, \dots, s_n wish to cross the bridge as quickly as possible. The rules remain:

- It is nighttime and you only have one flashlight.
- A maximum of two people can cross at any one time
- Any party who crosses, either 1 or 2 people must have the flashlight with them.
- The flashlight must be walked back and forth, it cannot be thrown, etc.
- A pair must walk together at the rate of the slower person's pace.

Give a linear time algorithm to find the fastest way to get a group of people across the bridge. You **must** have a proof of correctness for your method.

† † †

15. You have n heterosexual men and n heterosexual women. Each man ranks the women in order of preference. Each woman ranks the men in order of preference. Consider the following incredibly stereotypical courting algorithm. On stage i , each man goes to pitch woo on the porch of the woman that he prefers most among all women that have not rejected him yet. At the end of the stage the woman rejects all the men on her porch but the one that she favors most. Note that a woman may not reject a man in some stage, but later end up rejecting that man if a better prospect arrives on her porch. If it should ever happen that there is exactly one man on each porch, the algorithm terminates, and each woman marries the man on her porch. (You may be interested to know that medical schools really use this algorithm to fill intern positions.)

- (a) Give an upper bound as a function of n of the number of stages in this algorithm.
- (b) A marriage assignment is stable if there does not exist a man x and a woman y such that x prefers y to his assigned mate, and y prefers x to her assigned mate. Clearly adultery is a risk if a marriage assignment is not stable. Prove that this algorithm leads to a stable marriage.
- (c) A stable marriage M is man optimal if for every man x , M is the best possible stable marriage. That is, in every stable marriage other than M , x ends up with a woman no more preferable to him than the woman he is married to in M . Prove or disprove the above algorithm produces a man optimal stable marriage.

- (d) A stable marriage M is woman optimal if for every woman y , M is the best possible stable marriage. That is, in every stable marriage other than M , y ends up with a man no more preferable to her than the man she is married to in M . Prove or disprove the above algorithm produces a woman optimal stable marriage.
- (e) A stable marriage M is man pessimal if for every man x , M is the worst possible stable marriage. That is, in every stable marriage other than M , x ends up with a woman no less preferable to her than the woman he is married to in M . Prove or disprove the above algorithm produces a man pessimal stable marriage.
- (f) A stable marriage M is woman pessimal if for every woman y , M is the worst possible stable marriage. That is, in every stable marriage other than M , y ends up with a man no less preferable to her list than the man she is married to in M . Prove or disprove the above algorithm produces a woman pessimal stable marriage.

† † †