

Solutions for Homework Assignment #1

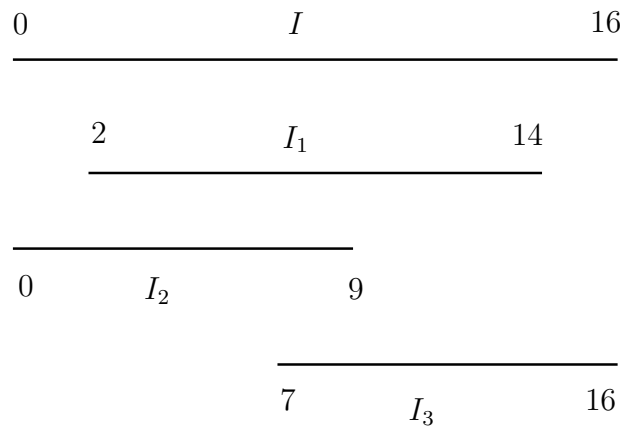
**Answer to Question 1.** The greedy algorithm that minimises the *maximum* lateness does not also minimise the *sum of latenesses*. A counterexample follows:

Job	Length	Deadline
<i>a</i>	3	1
<i>b</i>	1	2

The greedy algorithm that minimises maximum lateness orders the jobs in increasing deadline. In this example, the algorithm orders *a* before *b*, resulting in lateness of 2 for job *a* and 2 for job *b*, so the sum of latenesses for the two jobs is 4. In the schedule where *b* is before *a*, however, job *b* has lateness 0 and job *a* has lateness 3, so the sum of latenesses of the two jobs is 3.

A clue that the algorithm that minimises maximum lateness does not also minimise the sum of lateness is provided by the proof of optimality for the former. A crucial step in that proof, where we showed that fixing an inversion does not increase the maximum lateness, does not apply to the sum of the latenesses.

**Answer to Question 2. a.** The figure below shows a counterexample.



In this case, the proposed greedy algorithm would start by putting  $I_1$  into the set of intervals kept, and would therefore be forced to include all three intervals  $I_1$ ,  $I_2$  and  $I_3$  to cover  $I$ , while the intervals  $I_2$  and  $I_3$  suffice.

**b.** We maintain a subset  $A$  of the intervals  $\{I_1, I_2, \dots, I_n\}$  that cover an initial segment of  $I = [S, F]$ , from  $S$  to some intermediate point  $C$ ,  $S \leq C \leq F$ . Initially  $A = \emptyset$  and  $C = S$ . In each stage we greedily expand  $A$  by an interval  $I_j$  whose left endpoint is  $\leq C$  and whose right endpoint extends as far to the right as possible, and we update  $C$  accordingly. The algorithm is shown below in pseudocode.

```

let  $I = [S, F]$  and  $I_i = [s_i, f_i]$  for  $i \in [1..n]$ 
 $A := \emptyset$ 
 $C := S$ 
while  $C < F$  do
    let  $j \in [1..n]$  be s.t.  $s_j \leq C$  and  $\forall k \in [1..n]$ , if  $s_k \leq C$  then  $f_j \geq f_k$ 
     $A := A \cup \{I_j\}$ 
     $C := f_j$ 
return  $A$ 
    
```

CORRECTNESS. We prove the correctness of this algorithm using the “promising set” approach. Specifically, we will prove that the set of intervals in  $A$  is always contained in some optimal set of intervals  $A^*$ . (We say that a subset of  $\{I_1, \dots, I_n\}$  is optimal if it covers  $I$  using the smallest number of intervals in that set.) The following invariants can be proved using a straightforward induction, which we omit:

- (a) If  $A \neq \emptyset$ , the set of intervals in  $A$  cover  $[S, C]$ .
- (b) If  $A \neq \emptyset$ , then  $C$  is the maximum right endpoint of all intervals in  $A$ .

We now prove the following

**Key Invariant.** *The set of intervals in  $A$  is contained in some optimal set of intervals.*

The proof is by induction on the iteration number. Let  $A_t$  and  $C_t$  be the values of variables  $A$  and  $C$ , respectively, at the end of iteration  $t$ . (By convention, “the end of iteration 0” refers to the point just before the first iteration of the loop.)

The basis is trivially true, since  $A_0 = \emptyset$ . Suppose the invariant holds after iteration  $t$ , and let  $A^*$  be an optimal set that contains  $A_t$ . We will now prove that the invariant remains true after iteration  $t + 1$  (if there is such an iteration). Let  $I_j = [s_j, f_j]$  be the interval added to  $A$  in iteration  $t + 1$ . That is,  $A_{t+1} = A_t \cup \{I_j\}$ . If  $I_j \in A^*$  then obviously  $A_{t+1} \subseteq A^*$  and we are done. So, suppose  $I_j \notin A^*$ . Then  $A^* - A_t$  must contain some interval  $I_{j'} = [s_{j'}, f_{j'}]$ ,  $j' \in [1..n]$ , such that  $s_{j'} \leq C_t$ . (If not, all intervals in  $A^* - A_t$  start strictly after  $C_t$  and, by Invariant (b) above, all intervals in  $A_t$  end no later than  $C_t$ . Therefore, the intervals in  $A^*$  don’t cover all of  $I$ , contradicting that  $A^*$  is an optimal set.) By the algorithm,  $I_j$  is such that  $f_j \geq f_{j'}$ . Therefore  $A^* - \{I_{j'}\} \cup \{I_j\}$  covers  $I$ ; furthermore this set has the same number of intervals as  $A^*$ , so it is also optimal. Finally,  $A_{t+1} = A_t \cup \{I_j\} \subseteq A^* - \{I_{j'}\} \cup \{I_j\}$ . So,  $A_{t+1}$  is contained in an optimal set, as wanted.

By the exit condition, at the end of the loop  $C \geq F$ , so by Invariant (a) above, the set of intervals in  $A$  covers  $I$ . By the Key Invariant, at the end of the algorithm, the set of intervals in  $A$  is contained in some optimal set. Since this set covers  $I$ , it *is* an optimal set. So, if the loop terminates, the algorithm returns an optimal set.

It remains to prove that the loop terminates. To see this, note that in each iteration of the loop an interval in  $\{I_1, \dots, I_n\}$  is added to  $A$ . So, if the loop does not terminate earlier, after  $n$  iterations all intervals will have been added to  $A$ . Since the entire set of intervals  $\{I_1, \dots, I_n\}$  covers  $I$ , by the time all intervals have been added to  $A$  (if not earlier), the intervals in  $A$  cover  $I$  and so, by Invariants (a) and (b), the loop terminates.

TIME COMPLEXITY. As just argued, the loop terminates after at most  $n$  iterations. In each iteration, we can find the next interval to add to  $A$  in  $O(n)$  time by scanning the given list of  $n$  intervals to find the one whose left endpoint is at most  $C$  and whose right endpoint is as large as possible. (By using cleverer data structures you learned in CSCB63, this can actually be done in  $O(\log n)$  time: We can store the intervals in the nodes of an augmented balanced search tree, using the left endpoint of the interval as the node’s key and also storing in each node the maximum right endpoint of all intervals in the subtree rooted at that node.) Thus the algorithm can be implemented in  $O(n^2)$  time (or, with the cleverer data structure, in  $O(n \log n)$  time).

**Answer to Question 3. a.** This strategy is not optimal, as the following counterexample shows. Suppose we have two canoeists, of heights 1 and 4, and two paddles of length 3 and 6.

The greedy strategy under consideration assigns the paddle of length 3 to the canoeist of height 4, and the paddle of length 6 to the canoeist of height 1. The average penalty of this assignment is  $\frac{1}{2}((4 - 3) + (6 - 1)) = 3$ . The other assignment, where the paddle of length 3 goes to the canoeist of height 1 and the paddle of length 6 goes to the canoeist of height 4 has average penalty  $\frac{1}{2}((3 - 1) + (6 - 4)) = 2$ . Thus, the greedy strategy under consideration is not always optimal.

b. This strategy always produces an optimal assignment. We will use the exchange method to prove this.

Recall that we represent an assignment of paddles to canoeists by a permutation  $\pi$  of  $1, 2, \dots, n$ , where  $\pi(i)$  is the paddle assigned to canoeist  $i$ . We define an *inversion* of such a permutation to be a pair  $i, j$  such that  $h_i < h_j$  but  $p_{\pi(i)} > p_{\pi(j)}$ . That is, an inversion corresponds to a situation where a shorter canoeist is given a longer paddle than a taller canoeist.

The greedy algorithm under consideration produces an assignment with no inversions. To prove that such an assignment is optimal, it suffices to prove the following two facts:

**Fact 1.** *Any two assignments that have no inversions have the same average penalty.*

**Fact 2.** *There is an optimal assignment that has no inversions.*

To prove Fact 1, note that two assignments that have no inversions can only differ in the assignment of paddles to canoeists of the same shoulder height, or the assignment to canoeists of paddles of the same length. It is immediate from the formula for the average penalty, that any two such assignments have the same average penalty. (The two assignments are essentially the same, by renaming canoeists of the same shoulder height and paddles of the same length.)

To prove Fact 2 we first show that, given any assignment with inversions we can reduce the number of inversions without increasing the average penalty (by swapping the paddles assigned to the canoeists involved in an inversion). By starting with any optimal assignment  $\pi$  and applying this result repeatedly, we obtain an assignment  $\pi^*$  with no inversions and average penalty no larger than the optimal assignment  $\pi$  we started with. Thus,  $\pi^*$  is an optimal assignment with no inversions.

It therefore only remains to prove the following fact.

**Fact 3.** *Suppose  $\pi$  is an assignment that has at least one inversion. There is an assignment  $\pi'$  that has fewer inversions than  $\pi$  and average penalty no greater than  $\pi$ .*

PROOF. Let  $pnlty(\pi)$  denote the average penalty of assignment  $\pi$ ; i.e.,  $pnlty(\pi) = \frac{1}{n} \sum_{i=1}^n |h_i - \ell_{\pi(i)}|$ .

Suppose  $\pi$  is an assignment that has at least one inversion, say involving canoeists  $i$  and  $j$ . Let  $\pi'$  be the assignment which is identical to  $\pi$  except that canoeists  $i$  and  $j$  exchange their paddles. More precisely,

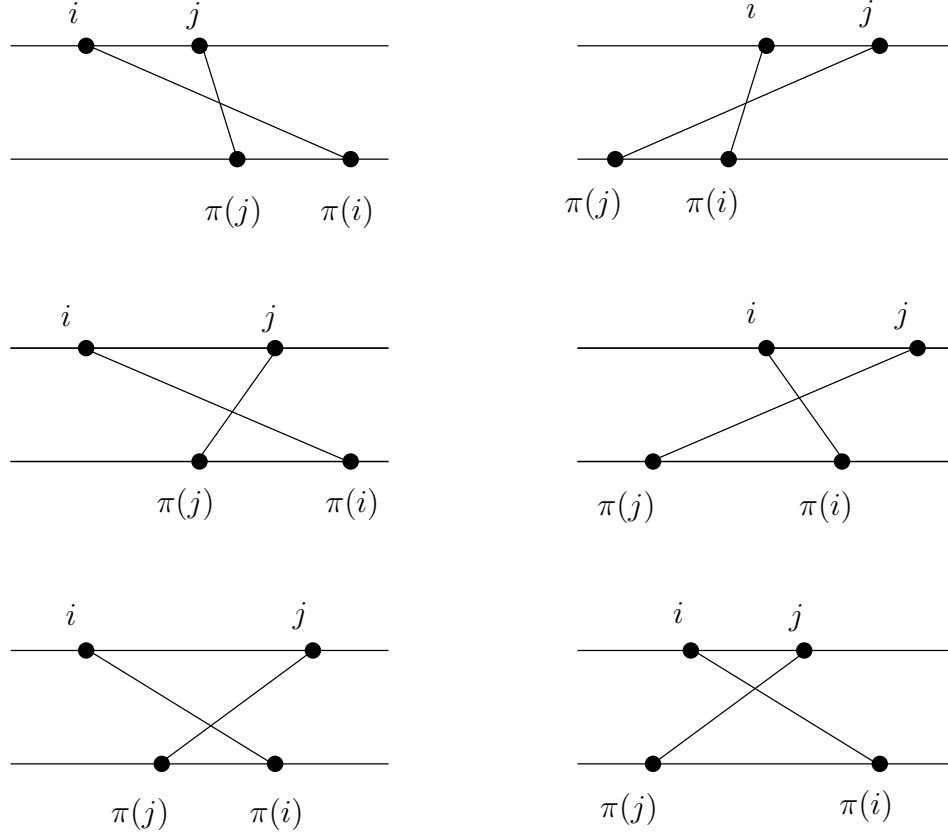
$$\pi'(k) = \begin{cases} \pi(k), & \text{if } k \notin \{i, j\} \\ \pi(j), & \text{if } k = i \\ \pi(i), & \text{if } k = j \end{cases}$$

Thus,

$$\begin{aligned} pnlty(\pi) - pnlty(\pi') &= (|h_i - \ell_{\pi(i)}| + |h_j - \ell_{\pi(j)}|) - (|h_i - \ell_{\pi'(i)}| + |h_j - \ell_{\pi'(j)}|) \\ &= |h_i - \ell_{\pi(i)}| + |h_j - \ell_{\pi(j)}| - |h_i - \ell_{\pi(j)}| - |h_j - \ell_{\pi(i)}| \end{aligned}$$

Note that  $\pi'$  has fewer inversions than  $\pi$ : We fixed at least one by exchanging the paddles of canoeists  $i$  and  $j$ . And if this exchange created in  $\pi'$  an inversion between some canoeist  $k$  and one of  $i$  and  $j$  that did not exist in  $\pi$ , then this very exchange also fixed an inversion between  $k$  and one of  $i$  and  $j$  that existed in  $\pi$  but no longer exists in  $\pi'$ .

It remains to show that  $pnlty(\pi) - pnlty(\pi') \geq 0$ . There are six cases to consider, depending on the order of  $h_i$ ,  $h_j$ ,  $p_{\pi(i)}$ , and  $p_{\pi(j)}$  relative to each other. These are illustrated in the figure below.



Each case on the right, however, is symmetric to the one on the left (simply by reversing all signs), and so it suffices to consider the three cases on the left.

CASE 1.  $h_i < h_j \leq \ell_{\pi(j)} < \ell_{\pi(i)}$ . In this case,

$$pnlty(\pi) - pnlty(\pi') = (\ell_{\pi(i)} - h_i) + (\ell_{\pi(j)} - h_j) - (\ell_{\pi(j)} - h_i) - (\ell_{\pi(i)} - h_j) = 0$$

CASE 2.  $h_i \leq \pi(j) \leq h_j \leq \ell_{\pi(i)}$ . In this case,

$$pnlty(\pi) - pnlty(\pi') = (\ell_{\pi(i)} - h_i) + (h_j - \ell_{\pi(j)}) - (\ell_{\pi(j)} - h_i) - (\ell_{\pi(i)} - h_j) = 2(h_j - \ell_{\pi(j)}) \geq 0$$

CASE 3.  $h_i \leq \pi(j) < \pi(i) \leq h_j$ . In this case,

$$pnlty(\pi) - pnlty(\pi') = (\ell_{\pi(i)} - h_i) + (h_j - \ell_{\pi(j)}) - (\ell_{\pi(j)} - h_i) - (h_j - \ell_{\pi(i)}) = 2(\ell_{\pi(i)} - \ell_{\pi(j)}) \geq 0$$

**Answer to Question 4.** Let  $T_1 = \{e_1, e_2, \dots, e_{n-1}\}$  and  $T_2 = \{f_1, f_2, \dots, f_{n-1}\}$  be two MSTs of  $G$ , where  $c(e_1) < \dots < c(e_{n-1})$  and  $c(f_1) < \dots < c(f_{n-1})$ . We will prove by complete induction that  $e_i = f_i$  for all  $i$  such that  $1 \leq i \leq n-1$ .

Let  $i$  be an arbitrary integer such that  $1 \leq i \leq n-1$ . Assume that, for all integers  $j$  such that  $1 \leq j < i$ ,  $e_j = f_j$ . We must prove that  $e_i = f_i$ .

Assume, for contradiction, that  $e_i \neq f_i$ . Since the edge costs are all distinct, we have  $c(e_i) \neq c(f_i)$ . Without loss of generality, assume  $c(e_i) < c(f_i)$ . Since  $e_i$  is not equal to any  $f_j$  for  $j < i$  (since for each such  $j$ ,  $e_j = f_j$ ), and since  $c(e_i) < c(f_i)$  and  $c(f_i) \leq c(f_j)$  for all  $j \geq i$ , we have that  $e_i \notin T_2$ .

So  $T_2 \cup \{e_i\}$  contains a (unique) cycle  $C$ . The edges of  $C$  cannot all be in  $T_1$ , since  $T_1$  is a tree. So  $C$  contains an edge  $f \in T_2 - T_1 \subseteq \{f_i, \dots, f_{n-1}\}$ . So  $c(e_i) < c(f_i) \leq c(f)$ . We now do the usual trick of

removing  $f$  from  $T_2 \cup e_i$ , creating the tree  $T_3 = (T_2 \cup \{e_i\}) - \{f\}$ .  $T_3$  has cost *smaller* than  $T_2$ , contradicting that  $T_2$  was minimal.

**Answer to Question 5.** In what follows, if  $T$  is a tree constructed during the execution of Huffman's algorithm,  $f(T)$  denotes the sum of the frequencies of the leaves of  $T$ . We refer to  $f(T)$  as the frequency of  $T$ .

**a.** An example is  $\Gamma = \{A, B, C, D\}$ , with  $f(A) = 2/5$  and  $f(B) = f(C) = f(D) = 1/5$ . One possible execution of Huffman's algorithm is to

- first create a tree  $T_1$  containing  $B$  and  $C$ , with frequency  $f(T_1) = 2/5$ ;
- then create a tree  $T_2$  containing  $A$  and  $D$ , with frequency  $f(T_2) = 3/5$ ; and
- finally create a tree  $T$  with subtrees  $T_1$  and  $T_2$ , with frequency  $f(T) = 1$ .

In the resulting tree, all codewords have length 2. Note that there is another possible execution of the algorithm resulting in a tree where  $A$  has a codeword of length 1.

**b.** Assume, for contradiction, that some symbol, say  $A$ , has frequency greater than  $2/5$ , yet Huffman's algorithm constructs a tree in which no codeword has length 1. Consider the step  $t$  when  $A$  is first merged with another tree  $T$ . (It is possible that  $T$  consists of a single node.) Let  $T_1$  be the tree that results from merging  $T$  and  $A$ .

How many trees other than  $T_1$  are there after step  $t$ ? There must be at least one such tree, say  $T_2$ : otherwise,  $t$  would be the last step of Huffman's algorithm and the codeword of  $A$  would have length 1. In every step we merge two trees of minimum frequency. Since one of the trees involved in step  $t$  was  $A$ ,  $f(T_2) \geq f(A) > 2/5$ . If after step  $t$  there was also a third tree  $T_3$  (in addition to  $T_1$  and  $T_2$ ), we would have  $f(T_3) > 2/5$  for exactly the same reason why  $f(T_2) > 2/5$ . But then  $f(T_1) + f(T_2) + f(T_3) > 6/5$ . This is impossible since at the end of each step, the frequencies of all trees must add up to 1.

Thus, after step  $t$  there are exactly two trees left,  $T_1$  and  $T_2$ . From this we conclude two facts:

- (i)  $f(T_1) + f(T_2) = 1$ . Since  $T_1$  resulted from merging  $T$  and  $A$ , we have  $f(T) = 1 - (f(A) + f(T_2)) < 1/5$ . Thus, at the start of step  $t$  there is a tree, namely  $T$ , of frequency less than  $1/5$ .
- (ii) The algorithm ends after step  $t + 1$ , producing a single tree by merging  $T_1$  and  $T_2$ . This implies that  $T_2$  does not consist of a single node: otherwise, the algorithm would produce a codeword of length 1. So,  $T_2$  has two subtrees, say  $T_{21}$  and  $T_{22}$ .

Consider now the step  $t' < t$  in which  $T_2$  was formed, by merging  $T_{21}$  and  $T_{22}$ . During that step there were certainly trees with frequency less than  $1/5$ . This is because as we just saw, at the start of a later step, namely  $t$ , there is still a tree, namely  $T$ , such that  $f(T) < 1/5$ . Thus, either  $T$  itself or a tree rooted at some node of  $T$ , of frequency even less than that of  $T$ , is available at step  $t'$ .

Since at each step we merge two trees of minimum frequency, and in step  $t'$  we merged  $T_{21}$  and  $T_{22}$ , it follows that the frequency of each of these trees is less than  $1/5$ . But then  $f(T_2) = f(T_{21}) + f(T_{22}) < 2/5$ , contradicting that, as previously shown,  $f(T_2) > 2/5$ .