case(II): e_i is accepted. We must show that $T \cup \{e_i\}$ is still promising. Since T is promising, there is a minimum cost spanning tree T_1 such that $T \subseteq T_1$.

subcase(a): $e_i \in T_1$. Then obviously $T \cup \{e_i\}$ is promising.

subcase(b): $e_i \notin T_1$. Then, according to the Exchange Lemma below, there is an edge e_j in $T_1 - T_2$ (where T_2 is the spanning tree resulting from the algorithm) such that $T_3 = (T_1 \cup \{e_i\}) - \{e_j\}$ is a spanning tree. Notice that $i < j$, since otherwise e_j would have been rejected from T and thus would form a cycle in T and so also in T_1. Therefore $c(e_i) \leq c(e_j)$, so $c(T_3) \leq c(T_1)$, so T_3 must also be a minimum cost spanning tree. Since $T \cup \{e_i\} \subseteq T_3$, it follows that $T \cup \{e_i\}$ is promising.

This finishes the proof of the Induction Step. □

Lemma 1.4 (Exchange Lemma) Let G be a connected graph, and let T_1 and T_2 be any two spanning trees for G. For every edge e in $T_2 - T_1$ there is an edge e' in $T_1 - T_2$ such that $T_1 \cup \{e\} - \{e'\}$ is a spanning tree for G.

Exercise 1.6 Prove this lemma. (**Hint:** let e be an edge in $T_1 - T_2$. Then $T_2 \cup \{e\}$ contains a cycle—can all the edges in this cycle belong to T_1?).

Exercise 1.7 Kruskal's Algorithm can return different spanning trees for the same input graph G, depending on how ties are broken when the edges are sorted into order (remember that before the algorithm proceeds, it orders the edges as follows: $c(e_1) \leq c(e_2) \leq \cdots \leq c(e_m)$, and breaks ties (i.e., edges of the same cost) arbitrarily). Show that for every minimum cost spanning tree T of a graph G, there exists a way of sorting the edges (i.e., breaking the ties) so that the algorithm returns T.

1.2 Job Scheduling with Deadlines and Profits

We have n jobs, each of which takes unit time, and a processor on which we would like to schedule them in as profitable a manner as possible. Each job has a profit associated with it, as well as a deadline; if the job is not scheduled by its deadline, then we don't get its profit. Because each job takes the same amount of time, we will think of a schedule S as consisting of a sequence of job "slots" $1, 2, 3, \ldots$, where $S(t)$ is the job scheduled in slot t.

Formally, the input to the problem is a sequence of pairs $(d_1, g_1), (d_2, g_2), \ldots, (d_n, g_n)$ where g_i is a non-negative real number representing the profit obtainable from job i, and $d_i \in \mathbb{N}$ is the deadline for job i.

Definition 1.4 A *schedule* is an array $S(1), S(2), \ldots, S(d)$ where $d = \max d_i$ (i.e., the latest deadline, beyond which no jobs can be scheduled), such that if $S(t) = i$, then job i is scheduled at time t, $1 \leq t \leq d$. If $S(t) = 0$, then no job is scheduled at time t.

Definition 1.5 A schedule S is *feasible* if (a) If $S(t) = i > 0$, then $t \leq d_i$ (i.e., every scheduled job meets its deadline), and (b) If $t_1 \neq t_2$ and $S(t_1) \neq 0$, then $S(t_1) \neq S(t_2)$ (i.e., each job is scheduled at most once).

Definition 1.6 Let the total profit of schedule S be $P(S) = g_{S(1)} + g_{S(2)} + \ldots + g_{S(d)}$, where $g_0 = 0$.

We want to find a feasible schedule S whose profit $P(S)$ is as large as possible. This can be accomplished with the following greedy algorithm:

Sort the jobs in non-increasing order of profits: $g_1 \geq g_2 \geq \ldots \geq g_n$
$d \longleftarrow \max_i d_i$
for $t : 1..d$
  $S(t) \longleftarrow 0$
end for
for $i : 1..n$
  Find the largest t such that ($S(t) = 0$ and $t \leq d_i$), and let $S(t) \longleftarrow i$
  [find the latest possible free slot meeting the deadline]
end for

Theorem 1.1 The greedy solution above is optimal (i.e., the profit $P(S)$ of the schedule S computed by this greedy algorithm is as large as possible).

Definition 1.7 A schedule is *promising* if it can be extended to an optimal schedule. Schedule S' extends schedule S iff for all $1 \leq t \leq d$, if $S(t) \neq 0$, then $S(t) = S'(t)$.

Example 1.1 If $S = (2, 0, 0, 0, 3)$, and $S' = (2, 0, 1, 0, 3)$, then S' extends S.

Lemma 1.5 The following is a loop invariant of the above greedy algorithm: *S is promising.*

Exercise 1.8 Why does Lemma 1.5 imply Theorem 1.1? (**Hint:** simple observation).

PROOF:(of Lemma 1.5) The proof is by induction.

Basis Case: Initially $S = (0, 0, \ldots, 0)$ is promising (take *any* optimal schedule; it is an extension).

Induction Step: Suppose that S is promising, and let S_{opt} be *some* optimal schedule that extends S. Let S' be the result of one more iteration through the loop where job i is considered. We must prove that S' continues to be promising, so the goal is to show that there is an optimal schedule S'_{opt} that extends S'. We consider two cases:

case (I) job i cannot be scheduled. Then $S' = S$, so we let $S'_{\text{opt}} = S_{\text{opt}}$, and we are done.

case (II) job i is scheduled at time t_0, so $S'(t_0) = i$ (whereas $S(t_0) = 0$) and t_0 is the latest possible time for job i in the schedule S.

subcase (a) job i is scheduled in S_{opt} at time t_1:

If $t_1 = t_0$, then, as in case (I), just let $S'_{\text{opt}} = S_{\text{opt}}$.

If $t_1 < t_0$, then let S'_{opt} be S_{opt} except that we interchange t_0 and t_1, that is we let $S'_{\text{opt}}(t_0) = S_{\text{opt}}(t_1) = i$ and $S'_{\text{opt}}(t_1) = S_{\text{opt}}(t_0)$. Then S'_{opt} is feasible [why (1)], it extends S' [why 2], and $P(S'_{\text{opt}}) = P(S_{\text{opt}})$ [why 3].

The case $t_1 > t_0$ is not possible [why 4].

subcase (b) job i is not scheduled in S_{opt}. Then we simply define S'_{opt} to be the same as S_{opt}, except $S'_{\text{opt}}(t_0) = i$. Since S_{opt} is feasible, so is S'_{opt}, and since S'_{opt} extends S', we only have to show that $P(S'_{\text{opt}}) = P(S_{\text{opt}})$. This follows from the following claim:

Claim: Let $S_{\text{opt}}(t_0) = j$. Then $g_j \leq g_i$.

We prove the claim by contradiction: assume that $g_j > g_i$ (note that in this case $j \neq 0$). Then job j was considered before job i. Since job i was scheduled at time t_0, job j must have been scheduled at time $t_2 \neq t_0$ (we know that job j was scheduled in S since $S(t_0) = 0$, and $t_0 \leq d_j$, so there was a slot for job j, and therefore it was scheduled). But S_{opt} extends S, and $S(t_2) = j \neq S_{\text{opt}}(t_2)$—contradiction.

This finishes the proof of the Induction Step. □

Exercise 1.9 Make sure you can answer all the "why's" in the above proof (the answers are given below for your reference, but you should first try without looking at them). Also, where in the proof of the claim you use the fact that $j \neq 0$?

Solution to the first part of exercise 1.9: [why 1] To show that S'_{opt} is feasible, we have to show that no job is scheduled twice, and no job is scheduled after its deadline. The first is easy, because S_{opt} was feasible. For the second we argue like this: the job that was at time t_0 is now moved to $t_1 < t_0$, so certainly if t_0 was before its deadline, so is t_1. The job that was at time t_1 (job i) has now been moved forward to time t_0, but we are working under the assumption that job i was scheduled (at this point) in slot t_0, so $t_0 \leq d_i$, and we are done.

[why 2] S'_{opt} extends S' because S_{opt} extended S, and the only difference is positions t_1 and t_0. They coincide in position t_0 (both have i), so we only have to examine position t_1. But $S(t_1) = 0$ since $S_{\text{opt}}(t_1) = i$, and S does not schedule job i at all. Since the only difference between S and S' is in position t_0, it follows that $S'(t_1) = 0$, so it does not matter what $S'_{\text{opt}}(t_1)$ is, it will extend S'.

[why 3] They schedule the same set of jobs, so they must have the same profit.

[why 4] Suppose $t_1 > t_0$. Since S_{opt} extends S, it follows that $S(t_1) = 0$. Since $S_{\text{opt}}(t_1) = i$, it follows that $t_1 \leq d_i$. But then, the algorithm would have scheduled i in t_1, not in t_0.

1.3 Huffman codes

Huffman[1] codes are a widely used and very effective technique for compressing data; savings of 20% to 90% are typical, depending on the characteristic of the data being compressed. In this section we consider the data to be a sequence of characters.

Huffman's greedy algorithm uses a table of the frequencies of occurrences of the characters to build up an optimal way of representing each character as a binary string.

Suppose we have a 100,000-character data file that we wish to store compactly. We observe that the characters in the file occur with the frequences given by the following table.

[1]This section comes from chapter 16.3 in *Introduction to Algorithms*, by Cormen, Leiserson, Rivest, and Stein.