1 Basic Definitions

A *Problem* is a relation from input to acceptable output. For example,

INPUT: A list of integers $x_1, \ldots, x_n$

OUTPUT: One of the three smallest numbers in the list

An algorithm A solves a problem if A produces an acceptable output for EVERY input.

A *optimization problem* has the following form: output a best solution S satisfying some property P. A best solution is called an *optimal solution*. Note that for many problems there may be many different optimal solutions. A *feasible solution* is a solution that satisfies the property P. Most of the problems that we consider can be viewed as optimization problems.

2 Proof By Contradiction

A proof is a sequence $S_1, \ldots, S_n$ of statements where every statement is either an axiom, which is something that we've assumed to be true, or follows logically from the precedding statements.

To prove a statement p by contradiction we start with the first statement of the proof as $\bar{p}$, that is not p. A proof by contradiction then has the following form

$$\bar{p}, \ldots, q, \ldots, \bar{q}$$

Hence, by establishing that $\bar{p}$ logically implies both a statement q and its negation $\bar{q}$, the only way to avoid logical inconsistency in your system is if p is true.

Almost all proofs of correctness use proof by contradiction in one way or another.

3 Exchange Argument

Here we explain what an exchange argument is. Exchange arguments are the most common and simpliest way to prove that a greedy algorithm is optimal for some optimization problem. However, there are cases where an exchange argument will not work.

Let A be the greedy algorithm that we are trying to prove correct, and $A(I)$ the output of A on some input I. Let O be an optimal solution on input I that is not equal to $A(I)$.

The goal in exchange argument is to show how to modify O to create a new solution O' with the following properties:

1. O' is at least as good of solution as O (or equivalently O' is also optimal), and

2. O' is "more like" $A(I)$ than O.

Note that the creative part, that is different for each algorithm/problem, is determininig how to modify O to create O'. One good heuristic to think of A constructing $A(I)$ over time, and then to look to made the modification at the first point where A makes a choice that is different than what is in O. In most of the problem that we examine, this modification involves changing just a few elements of O. Also, what "more like" means can change from problem to problem. Once again, while this frequently works, there's no guarantee.

4 Why an Exchange Argument is Sufficient

We give two possible proof techniques that use an exchange argument. The first uses proof by contradiction, and the second is a more constructive argument.

Theorem: The algorithm A solves the problem.

Proof: Assume to reach a contradiction that A is not correct. Hence, there must be some input I on which A does not produce an optimal solution. Let the output produced by A be $A(I)$. Let O be the optimal solution that is most like $A(I)$.

If we can show how to modify O to create a new solution O' with the following properties:

1. O' is at least as good of solution as O (and hence O' is also optimal), and

2. O' is more like $A(I)$ than O.

Then we have a contradiction to the choice of O.

End of Proof.

Theorem: The algorithm A solves the problem.

Proof: Let I be an arbitrary instance. Let O be arbitrary optimal solution for I. Assume that we can show how to modify O to create a new solution O' with the following properties:

1. O' is at least as good of solution as O (and hence O' is also optimal), and

2. O' is more like $A(I)$ than O.

Then consider the sequence $O, O'', O''', O'''', \ldots$

Each element of this sequence is optimal, and more like $A(I)$ than the proceding element. Hence, ultimately this sequence must terminate with $A(I)$. Hence, $A(I)$ is optimal.

End of Proof.

I personally prefer the proof by contradiction form, but it is solely a matter of

personal preference.

5 Proving an Algorithm Incorrect

To show that an algorithm A does not solve a problem it is sufficient to exhibit one input on which A does not produce an acceptable output.

6 Maximum Cardinality Disjoint Interval Problem

INPUT: A collection of intervals $C = \{(a_1, b_1), \ldots, (a_n, b_n)\}$ over the real line.

OUTPUT: A maximum cardinality collection of disjoint intervals.

This problem can be interpretted as an optimization problem in the following way. A feasible solution is a collection of disjoint intervals. The measure of goodness of a feasible solution is the number of intervals.

Consider the following algorithm A for computing a solution S:

1. Pick the interval I from C with the smallest right endpoint. Add I to S.

2. Remove I, and any intervals that overlap with I, from C.

3. If C is not yet empty, go to step 1.

Theorem: Algorithm A correctly solves this problem.

Proof: Assume to reach a contradiction that A is not correct. Hence, there must be some input I on which A does not produce an optimal solution. Let the output produced by A be $A(I)$. Let O be the optimal solution that has the most number of intervals in common with $A(I)$.

First note that $A(I)$ is feasible (i.e. the intervals in $A(I)$ are disjoint).

Let X be the leftmost interval in $A(I)$ that is not in O. Note that such an interval must exist otherwise $A(I) = O$ (contradicting the nonoptimality of $A(I)$), or $A(I)$ is a strict subset of O (which is a contradiction since A would have selected the last interval in O).

Let Y be the leftmost interval in O that is not in $A(I)$. Such an interval must exist or O would be a subset of $A(I)$, contradiction the optimality of O.

The key point is that the right endpoint of X is to the left of the right endpoint of Y. Otherwise, A would have selected Y instead of X.

Now consider the set $O' = O - Y + X$.

We claim that:

1. O' is feasible (To see this note that X doesn't overlap with any intervals to its left in O' because these intervals are also in $A(I)$ and $A(I)$ is feasible. And

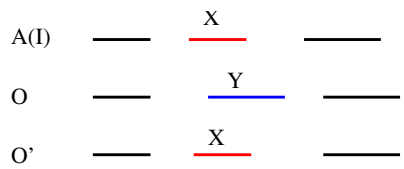

Figure 1: The instances $A(I)$, O and O'

X doesn't overlap with any intervals to its right in O' because of the key point above and the fact that O was feasible.),

2. O' has as many intervals as O (and is hence also optimal), and

3. O' has more intervals in common with $A(I)$ than O.

Hence, we reach a contradiction.

End of Proof.

7 Scheduling with Deadlines

The input consists of a collection of jobs represented by

$$(a_1, p_1, b_1) \dots (a_n, p_n, b_n)$$

of nonnegative integers. Here a_i is the release time, p_i is the processing time, and b_i is the deadline of job i. The output should be a schedule that assigns job i p_i unit time slots between time a_i and time b_i, if such a schedule exists.

The Earliest Deadline First (EDF) Algorithm schedules times one at a time from the earliest time to the latest time, at at each time runs the job, that has been released but not finished, with the earliest deadline. So the schedule is preemptive, in that the times when a job runs may not be contiguous.

Theorem: EDF is correct.

Proof by exchange argument: Assume to reach a contradiction that there exists an input X on which EDF does not produce an acceptable output. Let $G(X)$ be the output of EDF on X. Let $OPT(X)$ be the acceptable output that agrees with EDF on X for as many decisions as possible, or equivalently agrees with the longest possible prefix of $G(X)$. Thus EDF on X schedules a job later than its deadline while $OPT(X)$ does not, as this is the only plausible way that EDF can mess up.

Let t be the first time where $G(X)$ and $OPT(X)$ differ on the job scheduled at that time. Let i be the job scheduled at time t in $G(X)$ and let j be the job scheduled at time t in $OPT(X)$. Let u be the next time after t where $OPT(X)$ schedules job i. Note that u must exists because t was the first point

of disagreement of $OPT(X)$ and $G(X)$, $G(X)$ hadn't finished job i by time t (and thus neither had $OPT(X)$), and $OPT(X)$ is an acceptable output.

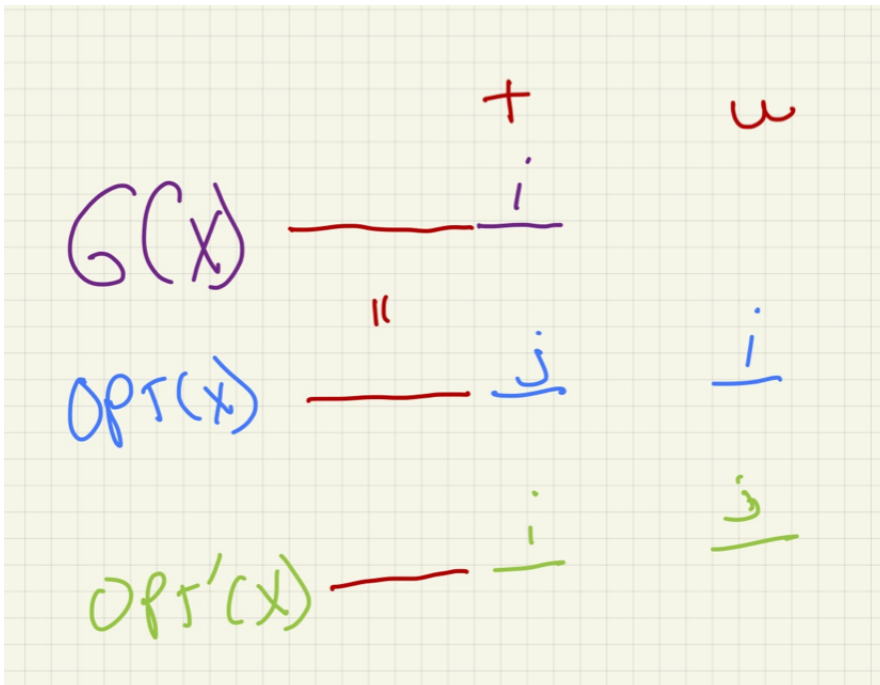

Figure 2: The instances $G(X)$, $Opt(X)$ and $Opt'(X)$

Let $OPT'(X)$ be equal to $OPT(X)$ except that i is scheduled at time t and j is scheduled at time u. Obviously $OPT'(X)$ agrees with $G(X)$ for at least one more time unit than does $OPT(X)$. $OPT'(X)$ is an acceptable output because:

- Each job is scheduled for the same amount of time as $OPT(X)$
- Job i can be feasibly run at time t because $G(X)$ runs it at time t, and EDF would not have run i at time t if that was before i's release time.
- Job j can be feasibly run at time u because:
  - Job i can be feasibly run at time u because $OPT(X)$ runs job i at time u, and $OPT(X)$ is an acceptable output.
  - Job's i deadline is earlier than job j's deadline because EDF selected to run job i at time t, when running job j at time t was feasible.

But then we reach a contradiction to the definition of $OPT(X)$ as $OPT'(X)$ is an acceptable output that agrees with $G(X)$ for one more step than does $OPT(X)$.

End of Proof.

8 Shortest Remaining Processing Time (SRPT)

The input consists of a collection of jobs represented by

$$(a_1, p_1) \dots (a_n, p_n)$$

of nonnegative integers. Here a_i is the release time, and p_i is the processing time. The output should be a schedule that assigns job i p_i unit time slots after time a_i so as to minimize $\sum_{i=1} c_i$, where the completion time c_i is the first time such that all p_i units of job i have been processed.

The Shortest Remaining Processing Time (SRPT) Algorithm schedules times one at a time from the earliest time to the latest time, at at each time runs the job, that has been released but not finished, with the least remaining units to process until the job will be completed. So the schedule is preemptive, in that the times when a job runs may not be contiguous.

Theorem: SRPT is correct.

Proof by exchange argument: Assume to reach a contradiction that there exists an input X on which SRPT does not produce an acceptable output. Let $G(X)$ be the output of SRPT on X. Let $OPT(X)$ be the acceptable output that agrees with SRPT on X for as many decisions as possible, or equivalently agrees with the longest possible prefix of $G(X)$.

Let t be the first time where $G(X)$ and $OPT(X)$ differ on the job scheduled at that time. Let i be the job scheduled at time t in $G(X)$ and let j be the job scheduled at time t in $OPT(X)$. Let U be the collection of unit time slots after t where $OPT(X)$ schedules either job i or job j. Note that during U that $OPT(X)$ schedules both jobs i and jobs j by the assumption that t is the first point of disagreement for $G(X)$ and $OPT(X)$.

Let $OPT'(X)$ be equal to $OPT(X)$ except during the unit time slots in U; During the time slots in U first i is run to completion and then j is run.

Obviously $OPT'(X)$ agrees with $G(X)$ for at least one more time unit than does $OPT(X)$ as $OPT'(X)$ runs i at time t. $OPT'(X)$ is an acceptable output because:

- Each job is scheduled for the same amount of time as $OPT(X)$
- Job i can be feasibly run at time t because $G(X)$ runs it at time t, and SRPT would not have run i at time t if that was before i's release time.
- Job j can be feasibly run at all times in U because $OPT(X)$ runs it at time t.
- The only jobs who completion times change are jobs i and j. The contribution of these jobs to the objective is $\min(c_i, c_j) + \max(c_i, c_j)$. But

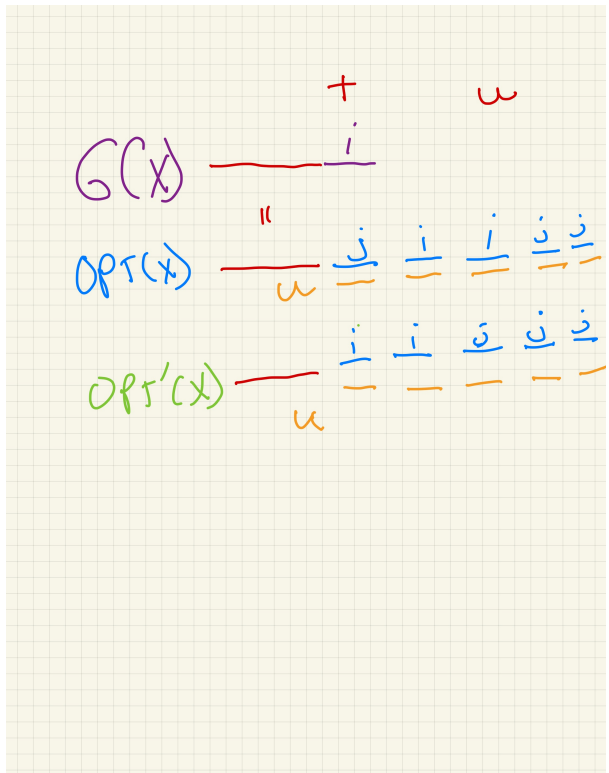

Figure 3: The instances $G(X)$, $Opt(X)$ and $Opt'(X)$

independently of when jobs i and j are run in U the value of $\max(c_i, c_j)$ is constant and does not change. Thus $\min(c_i, c_j) + \max(c_i, c_j)$ is minimized when $\min(c_i, c_j)$ is minimized, which obviously occurs when the job with the least remaining processing time left at time t is run to completion first. Note that the remaining processing time of job i at time t is no more than the remaining processing time of job j at time t by the definition of SRPT. Thus the total completion time for $OPT'(X)$ is at most the total completion time of $OPT(X)$.

But then we reach a contradiction to the definition of $OPT(X)$ as $OPT'(X)$ is an acceptable output that agrees with $G(X)$ for one more step than does $OPT(X)$.

End of Proof.

9 Kruskal's Minimum Spanning Tree Algorithm

We show that the standard greedy algorithm that considers the jobs from shortest to longest is optimal. See section 4.1.2 from the text.

Lemma: If Kruskal's algorithm does not included an edge $e = (x, y)$ then at the time that the algorithm considered e, there was already a path from x to y in the algorithm's partial solution.

Theorem: Kruskal's algorithm is correct.

Proof: We use an exchange argument. Let K be a nonoptimal spanning tree constructed by Kruskal's algorithm on some input, and let O be an optimal tree that agrees with the algorithms choices the longest (as we following the choices made by Kruskal's algorithm). Consider the edge e on which they first disagree. We first claim that $e \in K$. Otherwise, by the lemma there was previously a path between the endpoints of e in the K, and since optimal and Kruskal's algorithm have agreed to date, O could not include e, which is a contradiction to the fact that O and K disagree on e. Hence, it must be the case that $e \in K$ and $e \notin O$.

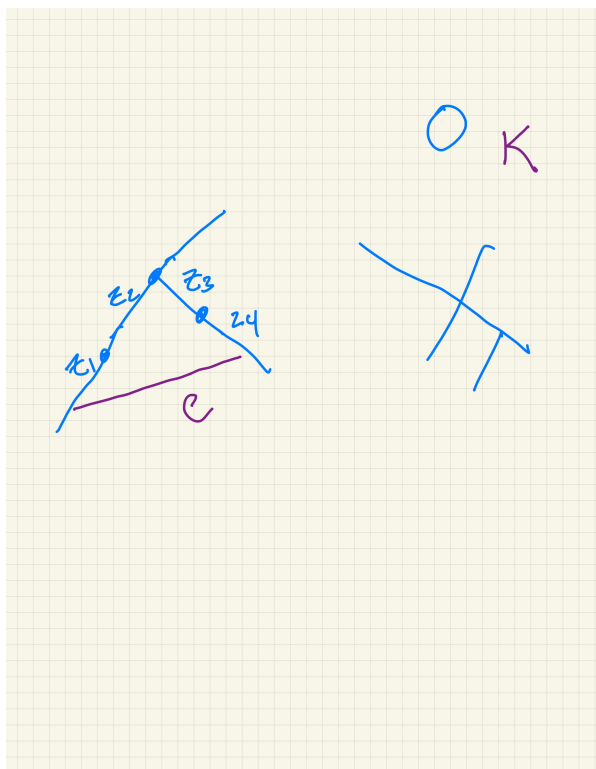

Figure 4: The instances $G(X)$, $Opt(X)$ and $Opt'(X)$

Let x and y be the endpoints of e. Let $C = x = z_1, z_2, \ldots, z_k$ be the unique cycle in $O \cup \{e\}$. We now claim that there must be an edge $(z_p, z_{p+1}) in C - \{e\}$ with weight not smaller than e's weight. To reach a contradiction assume otherwise, that is, that each edge $(z_i, z_{i+1}$ have weight less than the weight of (x, y). But then Kruskal's considered each $(z_i, z_{i+1}$ before (x, y), and by the choice of (x, y) as being the first point of disagreement, each (z_i, z_{i+1}) must be in K. But this is then a contradiction to K being feasible (obviously Kruskal's algorithm produces a feasible solution).

We then let $O' = O + e - (z_p, z_{p+1})$. Clearly O' agrees with K longer than O does (note that since the weight of (z_p, z_{p+1}) is greater than weight of e, Kruskal's considers (z_p, z_{p+1}) after e) and O' has weight no larger than O's weight (and hence O' is still optimal) since the weight of edge (z_p, z_{p+1}) is not smaller than the weight of e.

EndProof

10 Huffman's Algorithm

We consider the following problem.

Input: Positive weights $p_1, \ldots, p_n$

Output: A binary tree with n leaves and a permutaton s on $\{1, \ldots, n\}$ that minimizes $\sum_{i=1}^{n} p_{s(i)} d_i$, where d_i is the depth of the ith leaf.

Huffman's algorithm picks the two smallest weights, say p_i and p_j, and gives then a common parent in the tree. The algorithm then replaces p_i and p_j by a single number $p_i + p_j$ and recurses. Hence, every node in the final tree is label with a probability. The probability of each internal node is the sum of the probabilities of its children.

Lemma: Every leaf in the optimal tree has a sibling.

Proof: Otherwise you could move the leaf up one, decreasing it's depth and contradicting optimality.

Theorem: Huffman's algorithm is correct.

Proof: We use an exchange argument. Let consider the first time where the optimal solution O differs from the tree H produced by Huffman's algorithm. Let p_i and p_j be the siblings that Huffman's algorithm creates at this time. Hence, p_i and p_j are not siblings in O. Let p_a be sibling of p_i in O, and p_b be the sibling of p_j in O. Assume without loss of generality that $d_i = d_a \leq d_b = d_j$. Let $s = d_b - d_a$. Then let O' be equal to O with the subtrees rooted at p_i and p_b swapped. The net change in the average depth is $kp_i - kp_b$.

Hence in order to show that the average depth does not increase and that O' is still optimal, we need to show that $p_i \leq p_b$. Assume to reach a contradiction that indeed it is the case that $p_b < p_i$. Then Huffman's considered p_b before it paired p_i and p_j. Hence p_a's partner in H is not p_i. This contradicts the choice

of p_i and p_j as being the first point where they differ.

Using similar arguments it also follows that $p_j \leq p_b$, $p_i \leq p_b$, and $p_j \leq p_a$. Hence, O' agrees with H for one more step than O did (note that O and H could no.

EndProof.