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

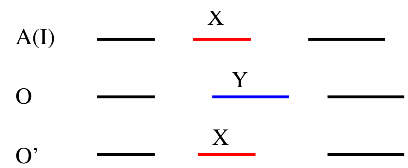

Figure 1: The instances $A(I)$, O and O'

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 Minimizing Total Flow time

We show that the Shortest Job First Algorithms is optimal for the scheduling Problem $1 \mid\mid \sum C_i$. A straight forward exchange argument is used.

Section 4.3.1 from the text.

8 Scheduling with Deadlines

We consider the problem $1 \mid r_i, \mid L_{\max}$. Each job J_k to be processed has an integer processing time p_k, an integer release time r_k, and an integer deadline d_k. A job may not be run before its release time, and you want to finish every job by its deadline. The Earliest Deadline First Algorithm schedules times one at a time from the earliest time to the latest time, at at each time runs the released job with earliest deadline. So the schedule is preemptive, in that the times when a job runs may not be contiguous.

Theorem: EDF will complete every job by its deadline if it is possible to do so.

Proof: Use an exchange argument. Consider an arbitrary collection of jobs $J_1, \ldots, J_n$, for which there is a schedule *Opt* that finishes all these jobs by their deadline. Let G be the greedy EDF schedule.

Assume that G and *Opt* agree on the first $k-1$ time steps, but run different jobs on the kth time step. Let the job run in G at time k be J_i and the job run at time k in *Opt* be J_j. Let ℓ be the next time after k that that J_i is run in *Opt*. The time ℓ must exist since *Opt* must finish J_i Now construct Opt' from *Opt* by running J_i at time k and J_j at time ℓ. So this moves J_i forward in time and J_j backward in time.

a) Obviously Opt' agrees with G for one more time unit than Opt dues.

b) We now argue that Opt' is still optimal by arguing that both J_i and J_i are run after their release time and before their deadline.

i) Since J_i moves forward in time in the swap, and since Opt completes all jobs by their deadlines, J_i completes by its deadline in Opt'. Since J_i is run at time k in G, and EDF won't run any job before its release date, Opt' can run J_i without worry that J_i is run before its release date.

ii) Since J_j was moved back in time in the swap, it is obviously run in Opt' after its release date.

KEY POINT: Since G runs J_i instead of J_j at time k then $d_i \leq d_j$ (since G and Opt agree for the first $k-1$ time units). Hence, time ℓ is before J_j's deadline since Opt runs J_i there, and $d_i \leq d_j$.

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 $e \notin O$.

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 $(z_p, z_{p+1}) inC - \{e\}$ 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.