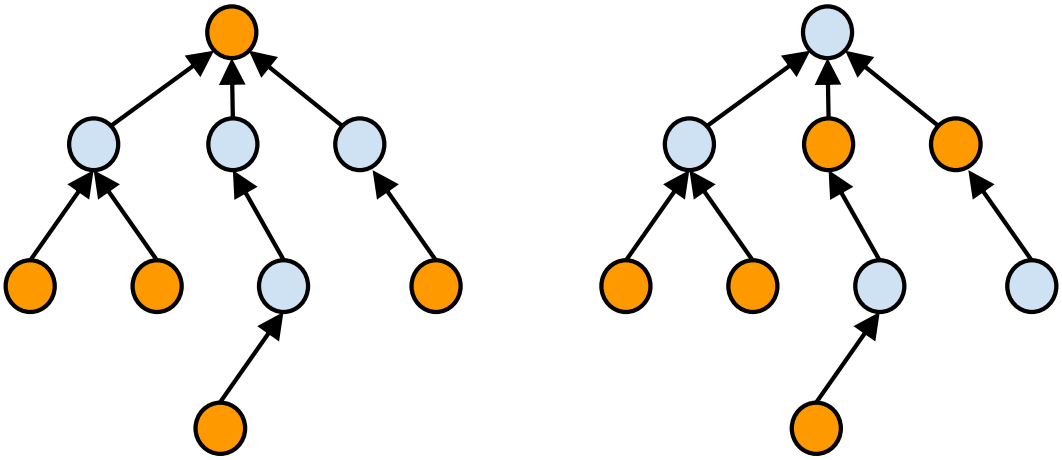
1. (6 points) Let $A\ \ (x) = a_0 + a_1 x$, let $B(x) = b_0 + b_1 x$, and let $C(x) = a_0 + b_0 x + a_1 x^2 + b_1 x^3$. If $A(9)\ = 15$ and $B(9) = -4$, what is the value of $C(3)$? What is the value of $C(-3)$?

This pertains to the Fast Fourier Transform. I see that A denotes the even terms of C and that B denotes the odd terms of C. I tried creating a butterfly network, but I couldn't make any sense of it. I know that for the FFT we took the even terms and added them to the odd terms for one of the answers, then we took the even terms minus the odd terms for the one on the other side of the unit circle. So even though I haven't fully convinced myself, I am going to say that

C(3) = A(9)+B(9) = 11
C(-3) = A(9)-B(9) = 19

2. Recall the *Maximum Independent Set Problem*. We are given a graph G and the task is to find a maximum sized set of vertices, no two of which belong to the same edge. In general, this task is NP-complete. In the special case where the graph is a tree, however, there is a polynomial algorithm. Two independent sets are shown as orange vertices in the tree below.



To keep notation consistent we will assume that the tree is rooted at a vertex r. We let parent(v) denote the vertex v's unique parent (thus, parent(5)=1), and we let children(v) be a list of vertices to whom v is a parent (thus, children(r)=[1,2,3]).

For any vertex v, we define $I(v)$ to be the size of the maximum independent set in the subgraph consisting of v and its descendents (its children and children's children, etc.)

   a. (6 points) Give a recurrence for $I(v)$ and argue for its correctness.

We want to consider whether it is better to include v in the independent set or if it is better to include a subset of the descendants of v.

Max{1 + I(children(children(v))), I(children(v))}

The first term comes from including vertex v in the set. Then we can't include any of the children of v, but we can include the grandchildren of v in the set. The second term comes from not including vertex v in the set and finding the maximum independent set of the children of v.

b. (6 points) Give an efficient algorithm for computing $I(r)$. (It's okay for your algorithm to be recursive. Just make it efficient.)

```
maxIndSet(G):
    # Base case: no children
    # All these vertices belong in the maximum independent set
    If children(r) == 0
        Return 1

    # Recursive case
    # Start at root of tree
    # Implement recurrence
    Return max( maxIndSet(children(r)), 1+maxIndSet(children(children(r))) )
```

c. (4 points) Give a runtime for your algorithm and justify your answer.

At the top level, each call to maxIndSet() would have to traverse a part of the tree. Since each vertex has exactly one parent, it looks like the tree is traversed at most twice (since there are two calls to maxIndSet). So it seems non-intuitive to me, but looks like it is O(V).

3. Suppose that you have found a maximum flow $f: E \to Z^+$in a network with integer capacities $c: E \to Z^+$. Unfortunately, after you have done your calculation, some damage has occurred to network link on the edge e, reducing the capacity to $c(e) - 1$.
    a. (2 points) Give an O(1) algorithm for testing whether the f is still a valid maximum flow.

Since you know what edge was damaged, you can simply check that the value of the flow going through edge e exceeds the newly diminished capacity. This can be done in constant time.

b. (6 points) Assuming that the old flow f is no longer valid, give an $O(|V| + |E|)$ algorithm for finding another maximum flow. Note that the new maximum flow may be $v(f)$ or $v(f) - 1$.

So we need to see if any of the edges upstream of the damaged edge have any residual capacity to take on the extra flow of the damaged edge. Depending on how far we have to re-route the flow, we also need to check the edged downstream. So basically we have to look at all

the edges to see if any of them have any residual capacity. So it isn't as easy as just going through all the vertices and finding which edges have residual capacities.

This isn't O(|E|+|V|), but it is better than nothing. I think you would want to start at the source of the damaged edge. Look at the other paths from that node and see which have residual capacities. If one of them does, than go down that path and continue looking for residual capacities. This sounds recursive, so it definitely wouldn't be O(|E|+|V|).

This is better, but still not O(|V|+|E|). For all the edges, add a value of 1 to the flow. This will take O(|E|) time and will take care of the 1 unit of flow we lost. Then, for each vertex, see if the conservation of flow holds (flow in = flow out) and that the capacities of the incoming edges are not exceeded. Reduce the flows by one for those whose capacities are exceeded and check that the conservation of flow is maintained. If it is still not conserved, we need to check the outgoing flows and see if any of those are exceeded. Reduce by one as necessary. If the conservation of flow is still not maintained,

c. (6 points) Argue why your algorithm for part b is correct.

So we know that the value of the flow cannot be more than v(f) (otherwise the original maximum flow wouldn't be maximum) and it can't be less than v(f) -1, keeping the same flow through the same paths, but with 1 less unit of capacity.

The second algorithm I present works by adding one to all the flows then checking that the flow is still valid. Since I am only changing the value of the initially valid flow along each edge by one, I only need to decrement the flow value by one when I go through to check. Adding one to all the flows "makes up" for the one unit we lost. I go through and make sure that the capacities are not exceeded and that the conservation of flow is maintained. Any flow that is left over should still be valid and it should at least handle v(f)-1.

4. (12 points) Consider the following cube solitaire puzzle. You are given a pile of n wooden cubes. Each cube has six faces, and there is a number between 1 and n written on each face. Your goal is to decide whether it's possible to arrange the cubes in a line so that the numbers on the faces pointing upward consist of the numbers $1,2,3, \ldots n$ in order. (Your choices consist both of how to order the cubes and also which face of each cube to choose as the face that points upward.) Such an arrangement is considered a solution of the puzzle. Give a polynomial-time algorithm that takes an instance of this cube solitaire puzzle and either produces a solution or reports (correctly) that no solution exists.

   Hint: Maximum Flow.

No clue.

5. One important class of linear programs are those of the form

minimize $\sum_{i=1}^{n} v_i$

subject to $v_i \geq b_i + \gamma \sum_{j=1}^{n} a_{ij} v_j$ for all i (1)

$v_i \geq b_i' + \gamma \sum_{j=1}^{n} a_{ij}' v_j$ for all i (2)

where the variables are $\{v_1, \ldots v_n\}$, and the constants have the properties that $0 \leq \gamma < 1$, $\sum_{j=1}^{n} a_{ij} = 1$, $\sum_{j=1}^{n} a_{ij}' = 1$ and $a_{ij}, a_{ij}' \geq 0$,

   a. (4 points) Introduce slack variables so as to change the inequalities for equalities.

Minimize $\sum_{i=1}^{n} v_i$

Subject to $v_i = w_i + b_i + \gamma \sum_{j=1}^{n} a_{ij} v_j$ for all i

$v_i = z_i + b_i' + \gamma \sum_{j=1}^{n} a_{ij}' v_j$ for all i

Since there are n inequalities for each constraint, we need to introduce n slack variables for each constraint. All the slack variables must be greater than or equal to zero.

   b. (8 points) Argue that if a finite optimum solution exists, it must have the property that $v_i = max\{b_i + \gamma \sum_{j=1}^{n} a_{ij} v_j, b_i' + \gamma \sum_{j=1}^{n} a_{ij}' v_j\}$.

In order to satisfy both constraints, v_i must be greater than or equal to the larger of the two constraints. See the numbered constraints above where the numbers refer to the right hand sides of the equations. In other words, $v_i \geq (1) \geq (2)$ or $v_i \geq (2) \geq (1)$ depending on which is greater, (1) or (2). This is the same as saying it is the maximum of the two constraints, which is what we wanted to prove.

6. Answer these two questions related to LP Duality.
   a. (4 points) If a linear program has no feasible solutions, what does that imply about the dual of the program?

If the primal program has no feasible solutions, than the dual is unbounded.

   b. (6 points) Find the dual of the following linear program

maximize $-x_1 + 3x_2 + 4x_3$

subject to $x_1 - 7x_2 + x_3 \leq 10$

$x_1 + 2x_2 - x_3 \leq 5$

Minimize $10y_1 + 5y_2$

Subject to $y_1 + y_2 \geq -1$

$-7y_1 + 2y_2 \geq 3$

$$y_1 - y_2 \geq 4$$

The maximization become minimization and the inequalities flip. The right hand side of the constraints in the primal becomes the coefficients of the objective function. The A matrix is transposed.

7. Given a graph $G = (V, E)$, a vertex cover is a set S of vertices so that every edge in E has at least one endpoint in S. Consider the following randomized algorithm for finding a small vertex cover.

$S = \emptyset$

for e in E
    if e is not covered by S
        choose an endpoint of e uniformly at random and add it to S.
return S

Note that the vertices within e are each chosen with probability $1/2$.

Let $U_i$be the random variable that is 1 when the ith edge is not covered when examined by the algorithm and that is 0 otherwise.

Let S^* denote a minimum cover (choose a particular one arbitrarily if there are more than one.). Let X_i be the random variable that is 1 if the vertex chosen from the ith edge is not in S^* and that is 0 otherwise.

We use the notation \ to indicate "set minus" ($A \backslash B = A \cap \underline{B}$).

a. (4 points) Express $E[|S|]$ in terms of $E[U_i]$ for $i \in \{1, \ldots m\}$ and express$E[|S \backslash S^*|]$in terms of $E[X_i]$. $i \in \{1, \ldots m\}$. What property of expectation are you using?

$$E[|S|] = \sum_{i=1}^{m} \frac{E[U_i]}{2}$$

Since each edge has two vertices each of which is equally likely to be in S.

$$E[|S \backslash S^*|] = \sum_{i=1}^{m}$$

b. (8 points) Show that $E[|S \backslash S^*|] \leq E[|S|]/2$.

c. (4 points) Show that $E[|S|] \leq 2|S^*|$.

Since each edge must have at least one vertex in S, and each edge has two vertices, there can be no more than two duplicates for a given vertex in S\S*. So the size of S cannot be more than twice the size of S*.

8. Suppose you are given a biased coin that comes up heads with probability p, but the value of p is unknown.

    a. (6 points) Give algorithm for generating an unbiased coin flip. That is to say, your algorithm should return "Heads" with probability ½ and "Tails" with probability ½. Hint: Consider a **pair** of coin flips and exploit symmetry.

What I first came up with:
Using the hint, given one coin flip we would want to output a coin flip that is opposite the current coin flip. So an algorithm that makes sense to me is to get the biased coin flip. Then, for the next coin flip return the opposite of the biased coin flip. For the next coin flip, use the biased coin again. And so on.

However this is wrong. This makes sure that the number of heads and tails is the same, but not that the probability of heads and probability of tails is the same for each flip. Take the case of P(H)=1, then we know that that every other flip will be heads and the other flips will be tails. This is not random.

It isn't explicit from the problem, but I assume I am supposed to use the biased coin flip in some way.

It seems that I would want to keep a running total of the current flips to continually estimate the value p. Then I would return the opposite of the flip with some probability based on my estimate of p. This is like what we did with conditional expectation with maintaining the invariant.

Given two coin flips with an unbiased coin, we know that there is a 50% probability that the flips will differ and there is a 25% chance each that the flips are the same (both heads or both tails). With a biased coin, the probability that we have two flips of the same value is p^2 for heads and $(1-p)^2$ for tails. The probability that they are different is p*(1-p). These quantities need to be kept proportional to the unbiased values in some way.

    b. (4 points) Give a proof for the correctness of your algorithm.

By construction this will have a head to tails ratio of 1:1 assuming that the algorithm is called an even number of times. The invariant is maintained, so that the number of heads and tails is always as close together as possible. This doesn't mean that the probability of each flip is 50% heads, 50% tails.

c. (4 points) What is the probability (in terms of p) that your algorithm requires more than 100 coin flips?

This is the one question that makes me think I have constructed my algorithm incorrectly. I only need two coin flips to have an unbiased coin flip. That is because what I had at first isn't really unbiased.