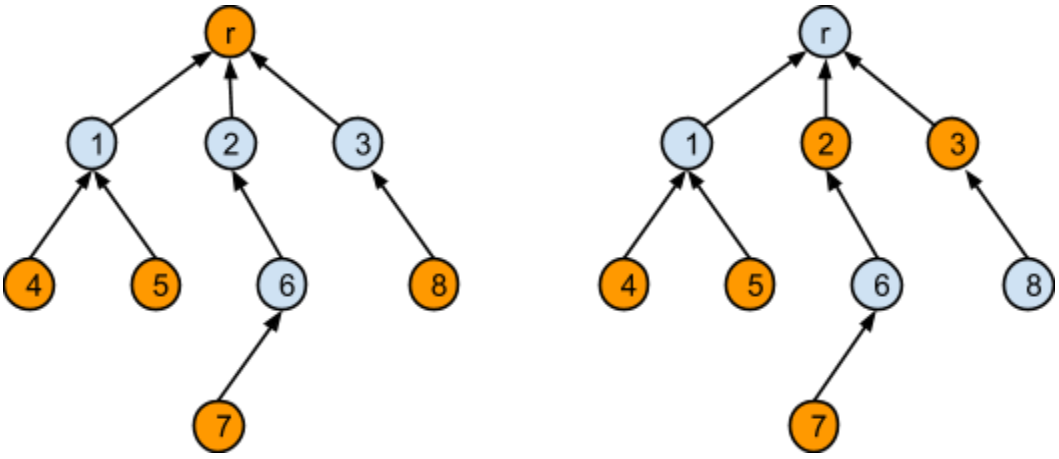
1. (6 points) Let $A(x) = a_0 + a_1 x$, let $B(x) = b_0 + b_1 x$, and let $C(x) = a_0 + b_0 x + a_1 x^2 + b_1 x^3$. If $A(9) = 15$ and $B(9) = -4$, what is the value of $C(3)$? What is the value of $C(-3)$?

Answer: Since $A(9) = 15$ and $B(9) = -4$, then we have $a_0 + 9a_1 = 15$, and $b_0 + 9b_1 = -4$. $C(3) = a_0 + 3b_0 + 9a_1 + 27b_1 = a_0 + 9a_1 + 3b_0 + 27b_1 = 15 + (-4) * 3 = 3$ .Similarly, $C(3) = a_0 - 3b_0 + 9a_1 - 27b_1 = 27$.

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 on 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.

Answer: the recurrence for $I(v)$ would be either v is included in the independent set or not.

Case 1: If v is included, all of its direct children are excluded, and check all of its grand children.

Case 2: If v is excluded, we will check all of its children.

Base Case: $I(v)$ = 1 if v is a leaf node, and $I(v)$ = 0 if v does not exist.

If $I(v) > 0$, return $I(v)$.

Otherwise,

$$I(v) = max(Sum(I(u)\ for\ u\ in\ children(r),\ \ Sum(I(w)\ for\ w\ is\ a\ grand\ children\ of\ v)\ +\ 1)\ .$$

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

Answer: We can combine with the the recurrence above and memorize all the $I(v)$ in an array.

function I(v):

1. Setting up an array A of length |V| to memorize the results, and initialize A[i] = -1 for all A[i] in A.
2. Set A[u] = 1 for all the leaf nodes u in graph G.
3. return I(v, A)

function I(v, A):

1. If v == null, return 0
2. if A[v] == -1, then A[v] = $max(Sum(I(u,\ A)\ for\ u\ in\ children(r),\ Sum(I(w,\ A)\ for\ w\ is\ a\ grand\ children\ of\ v)\ +\ 1)\ )$
3. return A[v]

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

Answer: the algorithm will have an complexity of O(n). Because for calculating I(v) for each node requires constant time as we memorize all the I(v) values for each node. And there are at most n times of calculation for I(v) as we have n nodes. Therefore the complexity is of O(n).

3. Suppose that you have found a maximum flow $f : E \rightarrow Z^+$ in a network with integer capacities $c : E \rightarrow Z^+$. 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.

Answer: we can do this by checking the endpoint, say v, of the edge e if its in flow is equal to its out flow. That is $f_{in}(v) = f_{out}(v)$. If it is equal, then f is still a valid maximum flow.

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

Answer: we can do the following

1. For the point that e is pointing from,
2. Exam if the point can push 1 extra flow to other points, if it can then check the point that have extra out flow capacity to include the 1 extra flow, return $v(f)$ if it can, if not check the points that have edges point it.
3. If still cannot push 1 extra flow, check the points that have edges point it.
4. Repeat 2 until all points in G are visited.
5. return $v(f)$ -1

The worst case is to run all the points in V, and look for all the edges for the extra flow. Therefore the complexity is $O(|V|+|E|)$.

c. (6 points) Argue why your algorithm for part b is correct.
Answer: because if it is looking for the extra capacity to include the 1 extra flow along the path to the ending point v of e. It looks through all the points that could possible have flow to v. If there are no such point, in those paths, then the reduced capacity can not be retrieved, and it will eventually go all the way back to the source, therefore it is complete and correct.

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.

Answer:

We could use a bipartite maximum flow problem as a subroutine to solve this problem. We can transform the problem by the following:

1. Set up s, and t.

2. Create n points on the left hand to represent the dices, and each has an edge pointing from s. (takes O(n))
3. Create n points on the right hand to represent the numbers on dices, and each has an edge going to t.(takes O(n))
4. For each dice, connect it to the ith point on the right hand side if it has i on one of its face. (takes O(n))
5. Run Ford-Furkson algorithm (takes $O(n^2)$)
6. If the max flow is equal to n, return true. Otherwise, return false.

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

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^{n} v_i \\ &\text{subject to} && v_i \geq b_i + \gamma \sum_{j=1}^{n} a_{ij} v_j \text{ for all } i \\ & && v_i \geq b_i' + \gamma \sum_{j=1}^{n} a_{ij}' v_j \text{ for all } i \end{aligned}$$

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.

Answer:

$$\begin{aligned} &\text{minimize} && \sum_{i=1}^{n} v_i \\ &\text{subject to} && v_i = b_i + \gamma \sum_{j=1}^{n} a_{ij} v_j + v_{n+i} \text{ for all } i \\ & && v_i = b_i' + \gamma \sum_{j=1}^{n} a_{ij}' v_j + v_{2n+i} \text{ for all } i \\ & && v_1, \ldots v_{3n} \geq 0 \end{aligned}$$

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

Answer: if a finite optimum solution exists, it must have the property that $v_i = \max\{b_i + \gamma \sum_{j=1}^{n} a_{ij} v_j,\ b_i' + \gamma \sum_{j=1}^{n} a_{ij}' v_j\}$. Suppose not, then there are two possibilities:

1. $v_i > \max\{b_i + \gamma \sum_{j=1}^{n} a_{ij} v_j,\ b_i' + \gamma \sum_{j=1}^{n} a_{ij}' v_j\}$
2. $v_i < \max\{b_i + \gamma \sum_{j=1}^{n} a_{ij} v_j,\ b_i' + \gamma \sum_{j=1}^{n} a_{ij}' v_j\}$

In the first case,

for each v_i we can find find another $v_i' = \max\{b_i + \gamma \sum_{j=1}^{n} a_{ij} v_j,\ b_i' + \gamma \sum_{j=1}^{n} a_{ij}' v_j\}$,

and therefore $v_i' < v_i$, so v is not the optimum solution.

In the second case,

It does not satisfy the constraints, therefore is a contradiction.

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?

      Answer: that implies that the dual of the program is unbounded according to Duality Theorem.

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

      maximize $-x_1 + 3x_2 + 4x_3$

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

      $x_1 + 2x_2 - x_3 \le 5$

      Answer: the dual problem of the program is

      min $10y_1 + 5y_2$

      subject to $y_1 + y_2 \ge -1$

      $-7y_1 + 2y_2 \ge 3$

      $y_1 - y_2 \ge 4$

      $y_1, y_2 \ge 0$

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 = ∅
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 $\setminus$ to indicate "set minus" ($A \setminus B = A \cap \overline{B}$).

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

Answer: $E[|S|]$ = Sum($E[P(U_i = 1)]$) for $i \in \{1, \ldots m\}$, and $E[|S \setminus S^*|]$ = Sum(1-$E[X_i]$) for $i \in \{1, \ldots m\}$.

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

Answer: Because for any vertex v in S^*, for any edge pointing to v, we have a probability of 0.5 to choose that vertex.

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

Answer: For any vertex v in S^*, for any edge pointing to v, we have a probability of 0.5 to choose that vertex. Therefore for each vertex v in S^*, there are at most 2 substitute in $E[|S|]$ if v is not in $E[|S|]$. Therefore $E[|S|] \leq 2|S^*|$.

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.

Answer: we can toss the coin twice to get the expected unbiased coin flip. We use "H" to represent a head, and "T" to represent a tail. Then the probabilities of a (H, T) and (T, H) in a pair of coin flips are equal which is $p * (1 - p)$. And we can use a (H, T) to represent a "new" head, and (T, H) to represent a "new" tail, other results are ignored.

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

Answer: as said in part a), the probabilities for both (H, T) and (T, H) are equal in the biased coin, which is $p(1-p)$. Therefore, they are equally possible. And they are the only results to be concerned. Therefore for the new unbiased coin, a head and a tail both have a probability of 0.5.

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

Answer: because each round requires 2 flips, and the probability that the result is a head or tail is $2p(1-p)$. Therefore the failure rate is $1-2p(1-p)$ for every two flips. There will be at 50 rounds that fail. So the probability is $(1-2p(1-p))^{50}$.