**Question 1** (Strongly Connected Components, 30 points). *Compute the strongly connected components of the graph below:*

Running DFS on the reverse graph we get the following pre and post orders: Running explore from A, we find the first component: \(ACG\). Next running explore from E (the next largest postorder number unused), we discover the component \(BEIKL\). Running explore on D, we find it is its own component. Finally, running explore from F yields the last component: \(FHJ\). So the components are: \(\{ACG\}, \{BEIKL\}, \{D\}, \{FHJ\}\).
Question 2 (Longest Common Subsequence, 30 points). Compute a longest common subsequence (the sequence, not just the length) of the strings

ACBABACB

and

BCACABAC.

We run the standard recurrence relation for computing the lengths of the longest common subsequences, getting the following table:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>B</th>
<th>A</th>
<th>B</th>
<th>A</th>
<th>C</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

The bolded entries here describe the path of entries actually used by the recurrence. Tracing them out we find the common subsequence of length 6: ACABAC. Note: this is in fact the unique longest common subsequence for this problem.
Question 3 (Tree Path Product, 35 points). Let $T$ be a rooted, balanced, binary tree with $n$ leaves and weights on the edges. Give an algorithm to compute the sum over all vertices of the product of the edge weights on the root to leaf path. For example, in the figure below the answer should be $2 \cdot 3 + 2 \cdot 4 + 1 \cdot 5 + 1 \cdot 6 = 25$. For full credit, your solution should run in $O(n)$ time.

![Tree Diagram]

We proceed by divide an conquer. We define $Prod(v)$ to be the sum over the leaves that descend from $v$ of the product of the weights along the path from $v$ to that leaf. We note that if $v$ has children $a$ and $b$ that $Prod(v) = w(v, a)Prod(a) + w(v, b)Prod(b)$, and that if $v$ is a leaf, we have $Prod(v) = 1$. From this we get the divide and conquer algorithm:

Prod($v$)
  If $v$ is a leaf
    Return 1
  Else
    Let $a$ and $b$ be the children of $v$
    Return $w(v, a) \cdot Prod(a) + w(v, b) \cdot Prod(b)$

Correctness is clear. For runtime, we note that each iteration does a constant amount of work and solves two smaller problems (one for each subtree). Since $T$ is balanced, each of these problems are approximately half the size of the original. Thus, we have the recurrence:

$$T(n) = 2T(n/2 + O(1)) + O(1),$$

which by the Master Theorem gives a runtime of $O(n)$. 
**Question 4** (Knapsack Counting, 35 points). Given $n$ items with positive integer weights $w_1, w_2, \ldots, w_n$ give an algorithm to compute the number of sets of these items (sets do not have repeated elements) with total weight at most $C$. For full credit, your algorithm should run in time $O(Cn)$ or better.

We proceed by dynamic programming. We let $Count(L, i)$ be the number of subsets of the first $i$ items with total weight at most $L$. We note that if $i = 0$ then $Count(L, i) = 1$. If $i > 0$, and $w_i > L$, then $Count(L, i) = Count(L, i - 1)$ (since the $i^{th}$ item cannot be used). If $i > 0$ and $w_i \leq L$, then $Count(L, i) = Count(L, i - 1) + Count(L - w_i, i - 1)$ as the first term counts the number of valid subsets without $w_i$ and the latter the number of subsets with. We can easily turn this into a dynamic program. Pseudocode is as follows:

```plaintext
Count(w_i, C)
Let A be an (C+1)x(n+1) array
For i = 0 .. n
  For L = 0 .. C
    If i = 0
      A[L,i] <- 1
    If i > 0 and w_i > L
    If i > 0 and w_i <= L
Return A[C,n]
```

Correctness is easily verified by the recursion. We claim by induction that when the entry $A[L,i]$ is filled in, it is filled with the correct value of $Count(L, i)$. This is because all other $A$ values referenced are correct by the inductive hypothesis and therefore the value assigned to $A[L,i]$ is that determined by the above recurrence.

To analyze the runtime, we note that there are $O(Cn)$ iterations through the inner loop, each of which takes constant time.
Question 5 (Pair Product Maximization, 35 points). Consider a list of real numbers \( x_1 < x_2 < \ldots < x_{2n} \). Stuart would like to split these numbers into \( n \) pairs so that the sum of the products of the paired numbers is as large as possible. For example, if the numbers were 1, 2, 3, 4, he could produce pairs (1, 3) and (2, 4) and get a total of \( 1 \cdot 3 + 2 \cdot 4 = 11 \).

Prove that Stuart maximizes his total by pairing \( x_1 \) with \( x_2 \), \( x_3 \) with \( x_4 \), and so on up to \( x_{2n-1} \) with \( x_{2n} \).

We proceed by induction on \( n \). Our base case is \( n = 1 \) for which the only pairing is to pair \( x_1 \) with \( x_2 \). Assuming that this is true for sequences of length \( 2n - 2 \) we will prove it for sequences of length \( 2n \). Firstly, we show that any optimal pairing must pair \( x_{2n-1} \) with \( x_{2n} \). This is because otherwise it will pair \( x_{2n} \) with \( x_i \) and \( x_{2n-1} \) with \( x_j \) for some \( i \neq j \) with \( i, j < 2n - 1 \). If instead of these pairings, we paired \( x_{2n} \) with \( x_{2n-1} \) and \( x_i \) with \( x_j \) (leaving all other pairs the same). This would increase the sum of products by

\[
\begin{align*}
  x_{2n}x_{2n-1} + x_ix_j - x_{2n}x_i - x_{2n-1}x_j &= (x_{2n} - x_j)(x_{2n-1} - x_i) > 0.
\end{align*}
\]

Thus, our optimal solution must pair \( x_{2n} \) with \( x_{2n-1} \). But the remainder of the pairing, must be an optimal pairing for \( x_1, x_2, \ldots, x_{2n-2} \), so by the inductive hypothesis, must pair \( x_1 \) with \( x_2 \), \( x_3 \) with \( x_4 \) and so on.
**Question 6** (Few Negative Edges, 35 points). Let $G$ be a weighted directed graph with $k$ negative weight edges, but no negative weight cycles. Given two vertices $s$ and $t$ in $G$ give an algorithm to compute the length of the shortest $s$ to $t$ path in $G$. For full credit, your algorithm should run in time $O(k^3 + k(|V| + |E|) \log |V|)$ or better.

Let $G'$ be the graph $G$ without the negative weight edges. Let $H$ be a new graph whose vertices are the endpoints of the negative weight edges of $G$ along with $s$ and $t$ (note that $H$ has $O(k)$ vertices). Let the edges of $H$ be the negative weight edges of $G$ plus for each pair of vertices $v, w$ in $H$ and edge from $v$ to $w$ of weight equal to the length of the shortest path from $v$ to $w$ in $G'$. We claim that the shortest path from $s$ to $t$ in $G$ has the same length as the shortest path from $s$ to $t$ in $H$. This is because any path from $s$ to $t$ in $G$ will consist of paths between vertices of $H$ using edges of $G'$ along with negative weight edges of $G$. Each such path will have a corresponding path in $H$ that is no longer than it. Likewise, any path in $H$ will have a corresponding path in $G$ that is no longer than it. Our algorithm proceeds as follows:

```
ShortestPaths(G,w,s,t)
    Find the vertices of H
    Construct G'
    For each vertex v of H
        Run Dijkstra(v) on G'
    For each vertex w of H
        Add an edge (v,w) to H with length given by the distance to w in the Dijkstra
    Add each negative weight edge of G to H
    Run Bellman-Ford on H to compute the distance from s to t and return the result
```

Note that this clearly correctly computes $H$, and Bellman-Ford computes the shortest path distance, and so by the above, this returns the right answer. To analyze the runtime, we run $O(k)$ Dijkstras in time $O(k(|V| + |E|) \log |V|)$, we add the edges of $H$ in $O(k^2)$ time, and the Bellman-Ford on $H$ runs in $O(|V_H||E_H|) = O(k \cdot k^2) = O(k^3)$ time.