1. (10 points) The figure below represents one of the trees in a dynamic trees data structure. Show the underlying representation of this tree (what Tarjan calls the virtual tree). Be sure to show the cost of each vertex using the differential form. Arrange each of the binary search trees so that the label of the node at the root of the BST comes first alphabetically, among all nodes in the BST.

In the analysis of dynamic trees, we defined an individual weight function on the vertices and a total weight function. What is the individual weight of $c$? What is the total weight of $e$?

$iw(c)=6, tw(e)=4$
2. (10 points) The correctness of any data structure operation depends on its maintaining certain essential properties of the data structure. The data portion of the class declaration for the C++ implementation of the balanced binary search tree data structure is shown below. What properties of the data must be maintained by programs that operate on it? List as many properties as you can.

```cpp
class bbsts {
    int n;   // trees defined on nodes {1,...,n}
    struct node {   // tree node structure
        int lchild, rchild, parent;
        int rank;
        keytyp keyfield;
    } *vec;
    ...
};
```

- \( n \) is non-negative and the number of nodes in the vec array is at least \( n+1 \).
- For \( 1 \leq u \leq n \), if \( \text{vec}[u].lchild \neq \text{Null} \) then \( 1 \leq \text{vec}[u].lchild \leq n \).
- For \( 1 \leq u \leq n \), if \( \text{vec}[u].rchild \neq \text{Null} \) then \( 1 \leq \text{vec}[u].rchild \leq n \).
- For \( 1 \leq u \leq n \), if \( \text{vec}[u].parent \neq \text{Null} \) then \( 1 \leq \text{vec}[u].parent \leq n \).
- For \( 1 \leq u \leq n \), \( \text{vec}[u].rank > 0 \).
- For \( 1 \leq u \leq n \), if \( \text{vec}[u].lchild = v \neq \text{Null} \) then \( \text{vec}[v].parent = u \). For \( 1 \leq u \leq n \), \( \text{vec}[u].rchild = v \neq \text{Null} \) implies that \( \text{vec}[v].parent = u \).
- The pointers must define a collection of trees. Specifically, this means that for \( 1 \leq u \leq n \), \( \text{vec}[u].lchild \neq \text{vec}[u].rchild \) and the parent pointers do not define any cycles.
- For \( 1 \leq u \leq n \), if \( \text{vec}[u].lchild = v \neq \text{Null} \) then \( \text{vec}[v].keyfield < \text{vec}[u].keyfield \).
- For \( 1 \leq u \leq n \), if \( \text{vec}[u].rchild = v \neq \text{Null} \) then \( \text{vec}[v].keyfield > \text{vec}[u].keyfield \).
- For \( 1 \leq u \leq n \), if \( \text{vec}[u].parent = v \neq \text{Null} \) then \( \text{vec}[u].rank \leq \text{vec}[v].rank \leq \text{vec}[u].rank + 1 \). If \( \text{vec}[v].parent = w \neq \text{Null} \) then \( \text{vec}[w].rank < \text{vec}[v].rank \). For \( 1 \leq u \leq n \), if \( \text{vec}[u].lchild = \text{Null} \) or \( \text{vec}[u].rchild = \text{Null} \), then \( \text{vec}[u].rank = 1 \).
3. (20 points) Suppose we modify the self-adjusting binary search tree so that it performs splay steps that make larger adjustments to the tree. In particular, suppose that for nodes that have at least 3 proper ancestors, we apply one of the four transformations shown below (or their mirror images).

How much closer does \( u \) get to the root during each of these splay steps? How much closer to the root do its descendants get during each splay step?

\( u \) gets 3 steps closer, and its descendants get either 2 steps or 3 steps closer.

Now, suppose we do a splay at some node \( s \) in such a self-adjusting binary search tree, and suppose that \( t \) is a descendant of \( s \) initially. Let \( \text{depth}(z) \) denote the distance to a node \( z \) from the root before the splay, and let \( \text{depth}'(z) \) denote the distance to \( z \) from the root following the splay. Give a tight upper bound on \( \text{depth}'(t) \) in terms of \( \text{depth}(s) \) and \( \text{depth}(t) \). Explain.

Each of the splay steps shown above causes descendants of \( s \) to two steps closer to the root, for every three steps closer that \( s \) gets. Consequently,

\[
\text{depth}'(t) \leq \text{depth}(t) - 2\left\lfloor \frac{\text{depth}(s)}{3} \right\rfloor
\]
How does your answer change if \( t \) is an ancestor of \( s \), initially? Explain.

In this case, the depth of \( t \) can increase by 3 during the splay step that makes \( t \) a descendant of \( s \), but then it will decrease. Consequently,

\[
depth'(t) \leq depth(t) + 3 - 2 \left\lfloor \frac{depth(t)}{3} \right\rfloor
\]

Do you think that it would be worthwhile modifying the implementation of self-adjusting binary search trees to allow the use of these larger splay steps? Why or why not?

It might be, but one would need to try it out and compare the two in order to be sure. The larger splay steps do seem to do a better job of balancing the tree, since they move vertices that are far from the root, closer to the root, than do the “standard” splay steps. On the other hand, there is some extra overhead associated with determining which of the different splay steps is applicable in a given situation. This could completely offset the speedup obtained by improving the balance.
4. (15 points) The differential cost representation used by the path sets data structure requires that $\Delta min$ and $\Delta cost$ be updated during a rotation operation. Show that the two equations below are correct.

$\Delta cost'(z) = \Delta cost(z) + \Delta min(z)$

$\Delta cost'(z) = cost'(z) - mincost'(z)$
  $= cost(z) - mincost(x)$
  $= cost(z) - (mincost(z) - \Delta min(z))$
  $= \Delta cost(z) + \Delta min(z)$

$\Delta min'(c) = \Delta min(c) + \Delta min(z)$

$\Delta min'(c) = mincost'(c) - mincost'(z)$
  $= mincost(c) - mincost(x)$
  $= (mincost(c) - mincost(z)) + (mincost(z) - mincost(x))$
  $= \Delta min(c) + \Delta min(z)$
5. (15 points) Consider an execution of the FIFO preflow-push algorithm on the network shown below. Show the state of the algorithm after the initialization step. You should show the initial distance labels, the flow for each edge with non-zero flow and the set of vertices that are in the queue. Also, mark all the edges that are not admissible by drawing a small line across them.

Suppose we divide the execution into passes, where a pass ends when the vertices that were on the queue at the start of the pass have all been removed from the queue. What vertices are on the queue at the end of the first pass?

\(a_2, a_6, e_1, b_2, b_6, b_{10}\)

What vertices are on the queue at the end of the second pass?

\(a_3, a_7, f_1, f_2, f_3, e_1, b_3, b_7, b_{11}\)

Identify any vertices that are relabeled during the first two passes and give their new distance label.

*The only vertex that is relabeled is \(e_1\), and its new label is 4.*
6. (10 points) The figure below shows an intermediate state in the execution of the minimum cost augmenting path algorithm for the min-cost max-flow algorithm, using vertex labels to represent transformed costs.

Draw the residual graph corresponding to this flow, and for each vertex, determine its distance from $s$ using the transformed costs. Write the distances by the vertices and mark the edges that are in the shortest path tree from $s$. Identify a minimum cost augmenting path in the graph. What is its cost, using the transformed costs?

The path $s$, $c$, $b$, $e$, $t$ is a minimum cost augmenting path with cost 1.

What is the new value of the vertex label for $e$ when this step is finished? For $t$?

The new distance label for $e$ is 2. The new distance label for $t$ is 1.