Be neat and concise, but complete.

1. (15 points) State the general greedy method for finding minimum spanning trees. Show that Prim’s algorithm is a special case of the general greedy method.

The general greedy method applies one of the following two coloring steps until all edges are colored. On completion, the blue edges form a minimum spanning tree.

- **Blue rule.** Find a cut crossed by no blue edges and at least one uncolored edge. Select a minimum weight uncolored edge crossing the cut and color it blue.

- **Red rule.** Find a cycle with no red edges and at least one uncolored edge. Select an uncolored edge of maximum weight from the cycle and color it red.

Prim’s algorithm builds a spanning tree incrementally. At each step, it selects a least-cost edge from the set of edges that has one endpoint in the tree. These edges are in the cut that separates the vertices that have been added to the tree from those that have not yet been added. Since the cut has no blue edges and at least one uncolored edge, each step is a correct application of the blue rule. The edges that are not in the tree when the algorithm ends are implicitly colored red. Since for each uncolored edge, the cycle consisting of the edge and the tree path joining its endpoints is a cycle that the red rule applies to, this implicit coloring is consistent with the general greedy method.
2. (15 points) A printout of a Fibonacci heap is shown below. Each set of values represents the (item number, key and rank). Mark bits that are set are shown as an asterisk.

```
  5 1 0
 7 2 2
  3 6 1
 12 19 0
 11 20 0
 8 3 3
 14 4 2*
  2 4 1
  4 9 0
  1 5 0
 15 8 1
  6 10 0
 10 5 1*
  9 13 0
```

Which items are siblings of item 10? **14 and 15**

What are the values of the four pointers associated with item 3 (identify each of the pointers)? *parent is 7, left sibling is 11, right sibling is 11 and child is 12.*

How many children did item 3 have when it first became a child of item 7? **1**

How many children did item 10 have when it first became a child of item 8? **2**

Suppose the item 2 is deleted from the heap. Modify the printout to show the effect of this.

*The modified printout appears below.*

```
  5 1 3
  4 9 0
 14 4 1
  2 4 1
  1 5 0
  7 2 2
  3 6 1
 12 19 0
 11 20 0
 8 3 2*
 15 8 1
  6 10 0
 10 5 1*
  9 13 0
```
3. (15 points) The picture below shows an implementation of a dynamic tree, using linked paths represented by binary search trees. Draw the actual tree that this represents (with vertex costs). Assuming the actual tree is being used in a maximum flow computation, what is the residual capacity of the path from $s$ to $t$ in the actual tree. Which edges become saturated if the maximum possible amount of flow is added to this path?

The actual tree appears at right. The edge from $b$ to $a$ is the only one that becomes saturated after 2 units of flow are added to the path $skfebadt$. 

![Diagram of the actual tree](image-url)
4. (15 points) How many changekey operations are performed by Dijkstra’s algorithm when executed on the graph shown below with vertex $v_1$ as the source vertex?

3 changekeys are performed when $v_2$ is removed from the heap, 2 are performed when $v_3$ is removed and 1 when $v_4$ is. So the total is 6.

Generalize this example to obtain an $n$ vertex graph on which Dijkstra’s algorithm performs $\Omega(n^2)$ changekey operations. Draw a diagram of this graph and explain why Dijkstra’s algorithm does require $\Omega(n^2)$ changekey operations.

The generalized graph has vertices $v_1$ to $v_n$ and edges from each $v_i$ to all vertices $v_j$ with $j>i$. For edges of the form $(v_i,v_{i+1})$, the weight is 1. For all other edges leaving $v_i$, the edge weight is $2(n-i-1)$. The graph is illustrated below.

When Dijkstra’s algorithm is run on this graph with $v_1$ as the source, the vertices are processed in numerical order. After vertex $v_i$ is processed, the distance to vertex $v_{i+1}$ is $i$ and the distance to vertices $v_j$ with $j>i+1$ is $(i-1)+2(n-i-1)=2n-i-3$. (This is easily proved by induction.) When vertex $v_{i+1}$ is processed, the distance to vertex $v_{i+2}$ changes from $2n-i-3$ to $i+1$ and the distance to vertices $v_j$ with $j>i+2$ changes to $i+2(n-i-2)=2n-i-4$. These changes in the distance result in changekey operations. The total number of changekey operations is $(n-2)+(n-1)+\ldots+1=(1/2)(n-2)(n-1)=\Omega(n^2)$. 
5. (15 points) Let $G$ be a directed graph and $t$ be a designated destination vertex. Let $T$ be a reverse shortest path tree with destination $t$. That is, $T$ is a spanning tree of $G$ directed toward $t$ in which all paths are shortest paths. Define $p(u)$ be the first vertex (after $u$) on the tree path from $u$ to $t$. Let $\text{secondBest}(u)$ be the first vertex (after $u$) on a second-best path from $u$ to $t$. More precisely, if $v$ is $\text{secondBest}(u)$, then $v \neq p(u)$, $(u,v)$ is an edge and there is no other such vertex $w \neq p(u)$, for which the shortest path from $u$ to $t$ that starts with $w$ is shorter than the shortest path from $u$ to $t$ that starts with $v$. Fill in the body of the C++ program shown below to compute $\text{secondBest}(u)$ for all vertices $u \neq t$. Your program should run in $O(m)$ time, where $m$ is the number or edges in $G$.

```c++
void secondBest(wdigraph G, vertex p[], int d[], vertex sb[]) {
    // Return the second best next hop vertex of u in sb[u] for all vertices u.
    // p[u] is the parent of u in the shortest path tree and d[u] is the length of the shortest path
    // from u to the destination.
    vertex u, v; edge e; int sblen
    for (u = 1; u != G.n; u++) {
        if (p[u] == Null) continue;
        sb[u] = Null;
        for (e = G.firstout(u); e!=Null; e = G.nextout(u,e)){
            v = G.head(e);
            if (v != p[u] &&
                (sb[u] == Null || G.w(e) + d[v] < sblen)) {
                sb[u] = v; sblen = G.w(e) + d[v];
            }
        }
    }
}
```
6. (15 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.

```c
class sbpartition {
    int n;   // partition is defined on {1,...,n}
    struct spnode {  // tree node
        int lchild, rchild, parent;
        int rank;
        keytyp keyfield;
    } *vec;
    ...
};
```

- $n$ is non-negative and for $n>0$, the number of nodes in the `vec` array is at least equal to $n+1$.

- For each vertex $u$, with `vec[u].lchild` not equal to Null, `vec[u].lchild = v` implies that `vec[v].parent = u`. Similarly, for each vertex $u$, with `vec[u].rchild` not equal to Null, `vec[u].rchild = v` implies that `vec[v].parent = u`.

- The pointers must define a collection of trees. Specifically, this means that the parent pointers cannot define any cycles.

- For each vertex $u$, with `vec[u].lchild` not equal to Null, `vec[u].lchild = v` implies that `vec[v].keyfield < vec[u].keyfield`.

- For each vertex $u$, with `vec[u].rchild` not equal to Null, `vec[u].rchild = v` implies that `vec[v].keyfield > vec[u].keyfield`.

- For each vertex $u$, with `vec[u].parent` not Null, `vec[u].parent = v` implies that `vec[u].rank ≤ vec[v].rank ≤ vec[u].rank + 1`. If `vec[v].parent` is not Null, `vec[v].parent = w` implies that `vec[u].rank < vec[w].rank`. For each vertex $u$ with fewer than two children, `vec[u].rank = 1`.
7. (10 points). The figure below represents an instance of the minimum cost, maximum flow problem. What is the cost of the flow that is shown? 46

Apply two steps of the minimum cost augmenting path algorithm. Show the changes resulting from the first step by modifying the first figure. Show the changes resulting from the second step by modifying the second figure.

The solution is shown in the second and third figures below, with changes highlighted.
8. (15 points) The figure below shows an intermediate state in the computation of the Edmonds-Karp algorithm for finding a maximum size matching in a general graph, with blossoms indicated by the shaded areas.

Draw a picture of the partition data structure associated with this state of the computation (there is no single right answer here, but your answer should be consistent with the figure).

For each blossom, identify its bridge, by marking it with the letter B.

What is the augmenting path found if edge \{g,l\} is examined next? \textit{rbjpdlgcehkfi}

List the edges in the matching, after the corresponding augmentation is performed. \{r,b\}, \{j,p\}, \{d,q\}, \{g,l\}, \{c,e\}, \{h,k\}, \{i,f\}, \{m,n\}