Uniform Cost Search (UCS) is an algorithm for traversing the nodes of a tree. UCS is similar to the breadth-first tree traversal algorithm we discussed in class; however, UCS uses a priority queue, instead of a queue, to order the nodes of the tree. In this assignment, you will implement Uniform Cost Search and test it by printing out the ancestors of George Washington in backwards order of their birth years. You will make use of a utilities package I have provided for you, which includes definitions of lists, trees, queues and priority queues.

BirthRecord Class

Define a BirthRecord class in the genealogy package. This class should be declared to implement the Comparable<BirthRecord> interface. Include two private instance variables: String name and int birthYear. Also define public methods String getName() and int getBirthYear() to access these variables. Also define an int compareTo(BirthRecord other) method to compare the year of this birth to the year of the other birth, returning -1, 0 or 1 according to whether this.birthYear is greater than, equal to or less than other.birthYear. Finally, define a String toString() method that returns a String composed of the name and the birthYear with a space in between them.

Genealogy Class

Modify the Genealogy class in the genealogy package. The Genealogy class extends ConsoleProgram and includes main and run methods to launch a console window. The run method demonstrates two ways of ordering the nodes of a binary tree. In order to complete this class, you must implement the following:

- Define an instance variable genealogy to be a BTree of BirthRecord objects. Initialize this variable to a tree that represents the ancestors of George Washington, including his parents and grandparents. You can find this information by googling: “George Washington genealogy”.

  public void queueOrderPrint(BTree<BirthRecord> bTree) This method should print out the nodes of the tree whose root is bTree, in order by generation, going backward in time, i.e., level by level in bTree. It should use the QueueListDL class included among the files for this assignment to control the order in which tree nodes are printed. This method is the essentially the same as the queueOrderPrint method we discussed in class, except that now bTree contains a BirthRecord in each node so the queue must work with BirthRecord objects.

  public void priorityQueueOrderPrint(BTree<BirthRecord> bTree) This method should print out the nodes of the tree whose root is bTree, in order according to decreasing value of birth year. For this purpose, this method should use the
PriorityQueueListSL class that you implemented in the lab for this week (included among the files for this assignment) to control the order in which tree nodes are printed.

BTree Class

Modify the BTree class in the utilities package. Recall that PriorityQueueListSL uses the compareTo method to order its contents. In order to use PriorityQueueListSL to order BTree nodes, you must modify the BTree<D> class in several ways:

- Declare the BTree class with the type parameter: <D extends Comparable<D>>, so that the type parameter D is required to implement the Comparable<D> interface. This means that D must include an int compareTo(D other) method that compares D type objects this and other. In our application, D will be BirthRecord, and you have already implemented compareTo for this class.
- Declare the BTree class so that BTree itself implements the Comparable<BTree<D>> interface. This means that BTree itself must implement an int compareTo(BTree<D> other) method that compares BTree<D> type objects this and other.
- Implement the int compareTo(BTree<D> other) method of the BTree class. This method should simply invoke compareTo on the data variable stored in this node and the other node.