University of Waterloo
Department of Electrical and Computer Engineering
ECE 250 – Algorithms and Data Structures

Final Examination (17 pages)
Instructor: Douglas Harder
April 14, 2004 9:00-12:00

Name (last, first)	Student ID

Do all problems.
If you are writing a supplemental exam, **DO NOT** attempt the question on Dijkstra's algorithm (I4). If you are writing the supplemental exam, circle the following:

WRITING SUPPLEMENTAL EXAM

The number in brackets denotes the weight of the question. If information appears to be missing from a problem, make a reasonable assumption, state it, and proceed. If the space to answer a question is insufficient, use the back of the previous page. You may use diagrams to supplement (but not replace) sentence answers.

Closed book. No calculators.

Question	Mark	Question	Mark
A	/9	F	/5
B	/7	G	/9
C	/6	H	/4
D	/12	I And Bonus (2)	/15 or /7*
E	/13	Total	/80 or /72*

* Total weight for students writing the supplemental exam.

A. Hash Tables

Every object in Java has a method with the signature `int hashValue()`. This method returns an integer which may be considered to be a hash of the object.

1. [7] Design a chained hash table class `ChainedHashTable` which implements the `HashTable` interface. Use the division method to convert the `int` returned by the `hashValue` method into an appropriate hash value. You may assume that the value returned by `hashValue()` is greater than or equal to 0. The argument `arraySize` indicates the number of entries in the array of the hash table. You should use the `LinkedLinked` class, the signature of which is given below.

Your implementation should assume that elements which are most recently inserted into the hash table are also the ones most likely to be referenced in the future.

```
public interface HashTable {
     public HashTable( int arraySize );
     public void insert( Object obj );
     public void extract( Object obj );
}

public class LinkedList {
     public LinkedList();
     public void append( Object obj );
     public void assign( LinkedList list );
     public void extract( Object obj );
     public Object getFirst();
     public LinkedList.Element getHead();
     public Object getLast();
     public LinkedList.Element getTail();
     public boolean isEmpty();
     public void prepend( Object obj );
     public void purge();
}
```

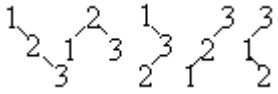
2. [2] Why is it appropriate that we can use the division method to map the integer returned by `hashValue()` onto the appropriate range?

B. Trees

1. [3] Taking into account only the instance variables of the various classes and their super classes, what is the memory requirement for a general tree with *n* nodes, as implemented and discussed in class. Recall that each `int` and object reference requires 4 bytes.

2. [4] Suppose data is stored in a binary search tree. For each of the four traversals, breadth-first traversal, and pre-, post- and, in-order depth-first traversals, determine if the order in which the data are visited depends on the order in which the data are stored.

For example, the data 1, 2, and 3 may be stored in a binary search tree in any one of the following five orders:



Which ones depend on the order in which the data are stored:

Which ones do not depend on the order in which the data are stored:

C. AVL Trees

[6] The tree shown in Figure 1 is an AVL tree of characters sorted using lexicographical (i.e., alphabetical) ordering. Starting with Figure 1, insert the letters A, C, E, and K **individually** into this tree and perform whatever rotations are necessary to keep the tree AVL balanced. For each step, indicate whether no rotations are necessary (i.e., the tree remains AVL balanced) or if a single rotation or a double rotation is necessary to rebalance the tree.

Warning: Do **not** add A, and then, into this result, add C, etc., that is, you should end up with four different trees, each with ten nodes.

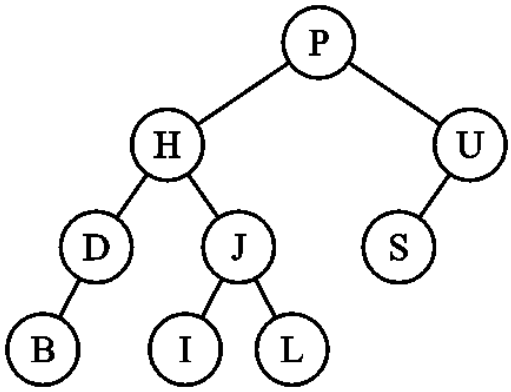


Figure 1.

D. Heaps

1. [4] For each of the following three parts (a, b, and c) the array represents a min-heap implemented as a complete binary tree. For each heap, either enqueue into or dequeue from the heap, as specified. The resulting representation should continue to be a complete binary tree. Each question is provided with a table to enter the resulting heap.

a. Enqueue the element 2 into the the heap represented by:

1	3	5	7	4	6	9								

b. Dequeue an element from the heap represented by:

1	3	5	7	4	6	9								

c. Dequeue an element from the heap represented by the following tree.

1	2	3	12	9	4	7	19	15	11	13	6			

2. [5] Assume you are implementing a priority queue using the data structures listed in the first column of Table 1. Given that there are *n* objects in the queue, fill in the asymptotic run times for the following methods:

	Enqueue	Dequeue	Get Head
Complete Binary Tree (Heap)	O( )	O( )	O( )
Ordered Linked list	O( )	O(1)	O(1)
Ordered Array	O( )	O(*n*)	O(1)
AVL Tree	O( )	O( )	O( )

Table 1.

One mark is taken off for each incorrect answer.

3. Suppose you are implementing the class `HeapAsArray` (again, using a complete binary tree) where the size of the array is both fixed with $2^n - 1$ entries. You have three ways of dealing with the problem of inserting an object into a full heap:

a. Throw an exception which is not a run-time exception.
b. Find the lowest value/priority object in the heap and replace it if the new object has a higher value/priority.
c. Discard the object being inserted.

i. [1] Which way has the slowest run time (a, b, or c)?

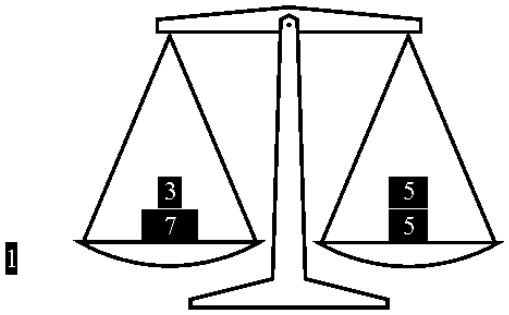
ii. [1] Which way forces the programmer using this class to write extra code to deal with a full heap (a, b, or c)?

iii. [1] If this class is to be used for a priority queue, which method (a, b, or c) will most likely be the cause of problems? If you think this question is ambiguous, you may justify your answer.

E. Backtracking Algorithms

Consider the following problem. Given a set of weights and a scale, we would like to find a combination of weights which is both **balanced** and **maximizes the weight on the scale**.

For example, given the weights 1, 3, 5, 5, and 7, placing the weights 7 and 3 on one scale, and the weights 5 and 5 on the other is both balanced and maximizes the weight (10) on the scale.



In this case, at any node of the decision tree if we have a balanced partial solution, we may treat this as a complete solution by assuming that all other weights may optionally be left off of either arm of the scale. Thus, before we start any problem, we already have one candidate for a potential solution: leave both arms of the scale empty, in which case, the scale is balanced. Denote the weight of the current optimal solution (initially 0) by the value m_{opt}.

1. [4] Write down the two mathematical equations (one for each restriction) which define the constraints of our problem. You may represent the weights by w_i and the set of weights on the left- and right-hand arms of the scale by L and R, respectively.

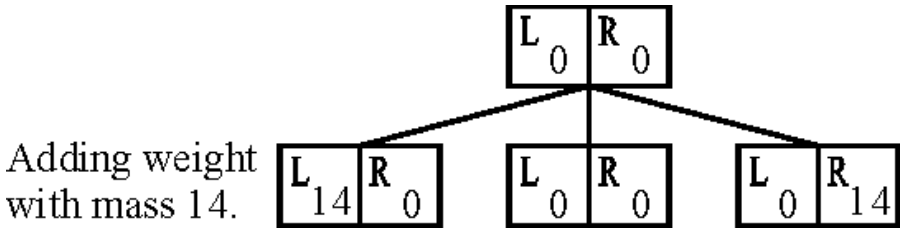
2. [1] One condition which allows us to prune a branch of the decision tree may be stated in English as "if the weight on one arm of the scale together with all unplaced weights are less than the weight on the other arm, then no solution can be found." Restate this mathematically. You may represent the set of all unplaced weights by W.

3. [2] There is another condition which depends on m_{opt}. State this condition mathematically for full marks, or in works for partial marks.

4. [6] Given five weights with masses 14, 5, 5, 2, and 1, draw the decision tree which finds the optimal solution to the problem described above. You need not draw any branches of the decision tree which are pruned by the conditions you give in parts 2 or 3 of this question. If you did not get part 3 of this question, do not despair; most pruning comes from the condition described in part 2. The first four nodes of the tree are given.

Hints:
a. Place the heaviest weights first, as is shown below.
b. You may use symmetry to eliminate some of the nodes.
c. The full decision tree has 364 nodes. Using the previous two hints and your conditions, the number of nodes which must be visited may be reduced to less than 30.

When you are pruning, place an X through the node which is being pruned and indicate if the pruning depends on the condition stated in part 2, part 3, or symmetry.



F. [5] Heap Sorting

Given the following integer (i.e., `int`) `MinHeap` and `MaxHeap` classes, implement the method `heapSort` which takes an array of integers and returns a new array of integers sorted in **ascending** order. Your answer should not be much more than five additional lines of code. The constructor argument `size` indicates the maximum size of the heap.

```
public class MaxHeap {
     public MaxHeap ( int size );
     public void enqueue( int x );
     public int dequeue();
}

public class MinHeap {
     public MinHeap ( int size );
     public void enqueue( int x );
     public int dequeue();
}

public int[] heapSort ( int[] list ) {
     int [] sorted =









     return sorted;
}
```

G. Divide-and-Conquer Sorting

1. [4] Sort the following data using a binary merge sort. Each step of subdividing and merging should occupy a separate line and subdivisions should be indicated by bold lines between the different lists. Continue subdividing until the size of the list is two, at which point you may simply order the two elements.

7	3	6	1	4	5	0	2

2. [2] What shortcoming of quick sort does merge sort avoid? Use one or two sentences.

3. [3] Is the reasoning given in the following statement correct, *yes* or *no*? If *yes*, explain why. If *no*, what is the problem with the author's reasoning?

> The standard binary merge sort takes $\log_2(n)$ steps to sort a list of n objects. If I divided the list into four sublists, sort each of these, and then merge these four sublists back into a single list, then this will take $\log_4(n)$ steps. Since it is true for all values of n that $\log_4(n)/\log_2(n) = ½$, only half as many steps will be required, and therefore this modified merge sort will be twice as fast as the standard binary merge sort.

H. Bucket Sort and Radix Sort

1. [4] Perform a increasing radix sort on

323, 312, 321, 112, 231, 123, 322

by using the following 3x5 tables, which may act as queues. For each step, enter the partial solution in the entry in the 1x7 table. The 6th table should be the sorted list.

1					
2					
3					

1					
2					
3					

1					
2					
3					

I. Graph Algorithms (and Bonus)

For parts 1, 2, and 3 of this question, you do not have to know anything about graphs to answer the questions.

Consider a directed graph (V, E) defined by a set of vertices V and a set of edges E, each edge being an ordered pair of vertices, e.g., (v_1, v_2). A path of length n is a nonempty ordered sequence of vertices (a_0, a_1, ..., a_n) such that each ordered pair (a_{i-1}, a_i) where i = 1, ..., n is an edge.

A path is simple if all the vertices are different with the one exception: the first and last verticies may be equal. Consider the following implementation of the class `Path` together with some additional classes.

```
public class Path {
      LinkedList path;
      int length;

      public Path( Vertex v ) {
            path = new LinkedList();
            path.append( v );
            length = 0;
      }
      public void append( Vertex v ) {
           if ( isEdge( v, path.getLast().getDatum() ) ) {
                  length++;
                  path.append( v );
            } else
                  throw new RuntimeException( "The new vertex is" +
                      "not adjacent to the last element the path" );
      };

      public boolean isSimple() {...}
}

public class Vertex {
      Object key;
      public Vertex( Object key ) { this.key = key; }
}

public class LinkedList {
      public LinkedList();
      public LinkedList.Element getHead();
      public Object getFirst();

      public class Element {
            public Element( Object key );
            public Element getNext();
            public Object getDatum();
      }
}
```

1. [5] Implement the method `isSimple` which returns `true` if the path is simple and `false` otherwise. You may, if you wish, use the class `ChainedScatterTable`, the signatures of which are given below. Your method must run in O(*n*) time where *n* is the length of the path. You may assume that two vertices with the same key will be equal when compared to each other.

```
public class ChainedScatterTable {
      public ChainedScatterTable( int length );
      Object find( Object obj );
      void insert( Object obj );
      boolean isFull();
      boolean isMember( Object obj );
      void purge();
      void withdraw( Object obj );
}

public class Path {
      ...
      public boolean isSimple() {









      }
}
```

Hints:
Implement code which ensures that all the edges are different and then add additional code to check for the special case that the first and last elements may be equal.
If you are significantly exceeding 10 lines of code, you should reconsider your strategy.

2. [2] Suppose instead that one additional instance variable, `boolean simple`, is added to the class `Path`. This instance variable is initially set to `true`. Each time another vertex is added to the path, the resulting path is tested and if it is no longer simple, this variable is set to `false`. In this case, the method `isSimple` simplifies to

```
public boolean isSimple() {
     return simple;
}
```

What must the asymptotic (big-O) run time of adding n vertices to the path be, assuming that the method `isEdge` runs in constant time?

3. **(Bonus)** [2] If, in addition to adding the variable `simple` (as described in part 2), suppose we add a chained hash table which stores all vertices in the path. In this case, it is very easy to test if the new vertex already occurs in the linked list and therefore the run time of adding n vertices into the path is $O(n)$. What rule (described in class) describing the relationship between run-time and memory used does this reflect?

4. Dijkstra's Algorithm

IF YOU ARE WRITING A SUPPLEMENTAL EXAM, DO NOT DO THIS QUESTION.

In finding the length of the shortest path from a given a vertex A in a graph to all other vertices in the directed graph the edges of which have positive weights, Dijkstra's algorithm is fast and efficient.

The vertex A is given the weight 0 and all other nodes are given a weight of infinity. Each node is marked as unvisited.

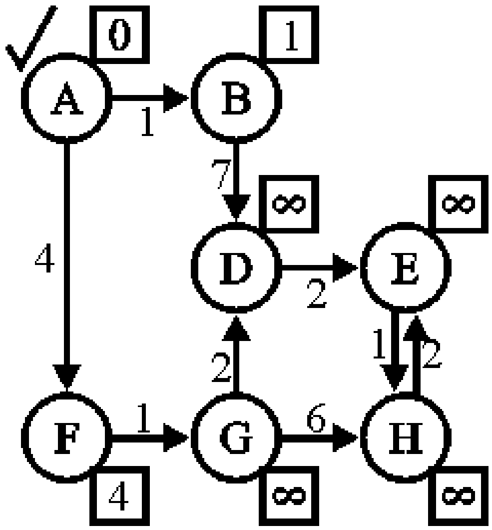
Select one of the unvisited vertices which have the smallest current weight. Mark this vertex as visited. For each unvisited vertex adjacent to this selected vertex, if the sum of the weight of the selected vertex and the edge joining the two vertices is less than the current weight of the adjacent vertex, update the weight of the adjacent vertex.

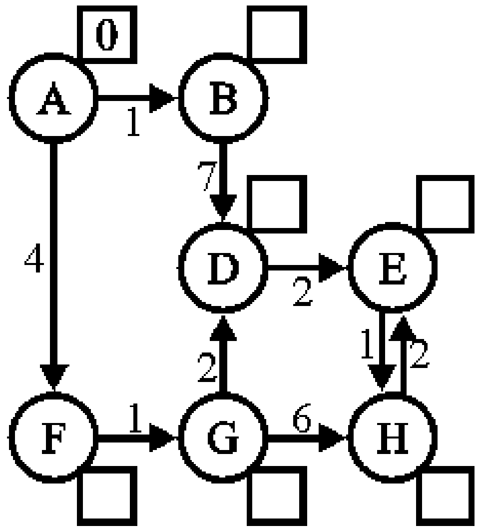
Repeat this process until all vertices have been marked as visited.

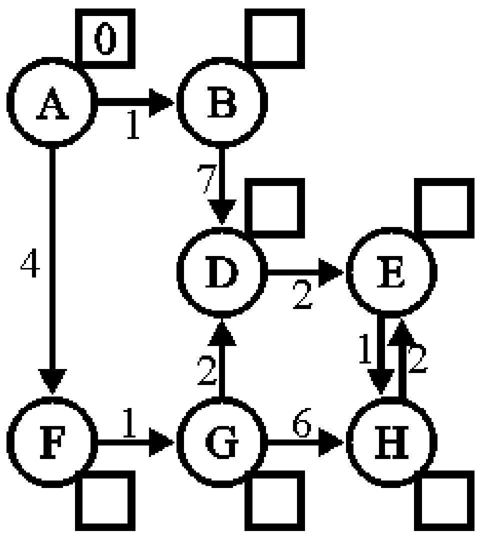
After all vertices have been visited, the weights on the vertices are the shortest distance to those vertices from the given vertex A.

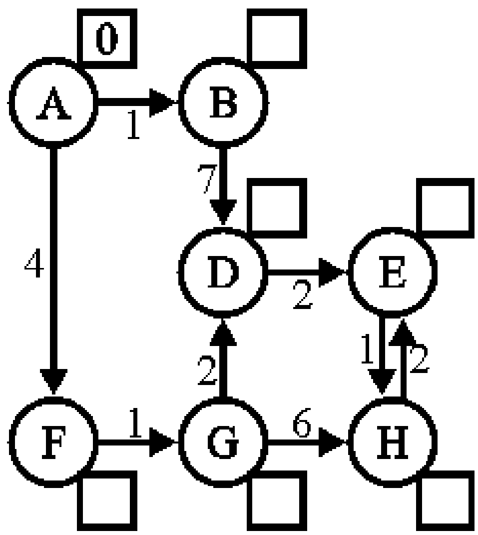
a. [6] Evaluate Dijkstra's algorithm on the directed graph described on the next page. The first iteration of the algorithm has already been performed. Use one image of the graph for each iteration.

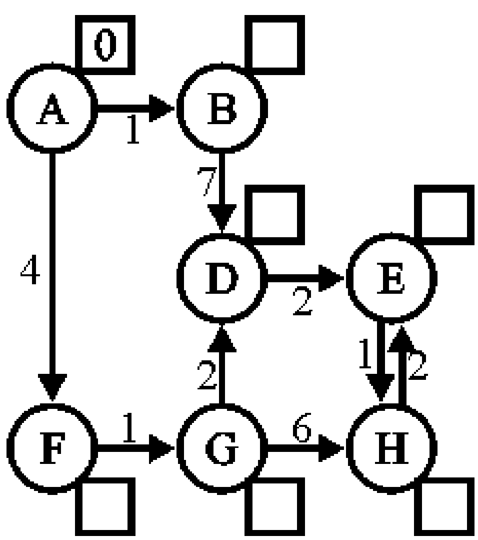
b. [2] Dijkstra's algorithm was described as being *greedy*. Underline the sentence in the description of Dijkstra's algorithm which classifies the algorithm as greedy.

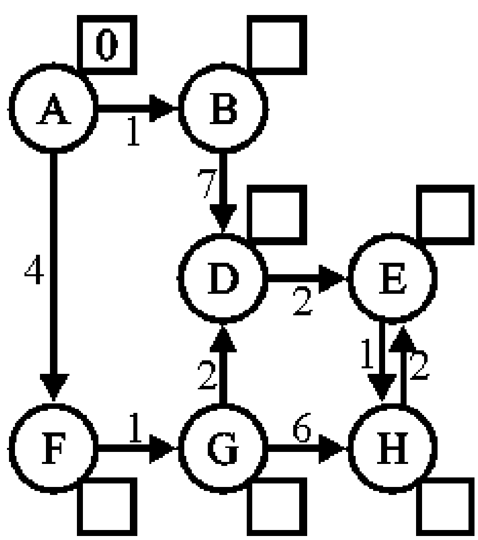

0
1
A
B
1
7
∞
∞
4
D
E
2
1
2
2
F
G
H
1
6
4
∞
∞


0
A
B
1
7
4
D
E
2
1
2
2
F
G
H
1
6


0
A
B
1
7
4
D
E
2
1
2
2
F
G
H
1
6


0
A
B
1
7
4
D
E
2
1
2
2
F
G
H
1
6


0
A
B
1
7
4
D
E
2
1
2
2
F
G
H
1
6


0
A
B
1
7
4
D
E
2
1
2
2
F
G
H
1
6