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

Final Examination
2005-12-14

Instructor: Douglas Wilhelm Harder
Time: 2.5 hours
Aides: none
15 pages

Surname, Given name	Student ID Number

You may only ask "May I go to the washroom?"
If you are unsure of a question, write down your assumptions and continue.
If you have insufficient space to answer a question, use the back of the previous page.
If you do not sign this page, you will get a bonus of one mark.
The examination is out of 70 marks.

I have read all the statements on this page.

_____

Signature

Marks.

1.1	2.1	2.2	2.3	2.4	3.1	3.2	3.3	4.1	4.2
4.3	4.4	4.5	5.1	5.2	6.1	6.2	7.1	7.2	8.1

1. **B-Trees**

1.1 **[5]** Figure 1a shows a B-tree. Perform the given three operations on this B-tree, though you need only draw those blocks which were modified as a result of the operation. Use only the basic merging or splitting of blocks where necessary. Do not use optimizations such as *adoption*.

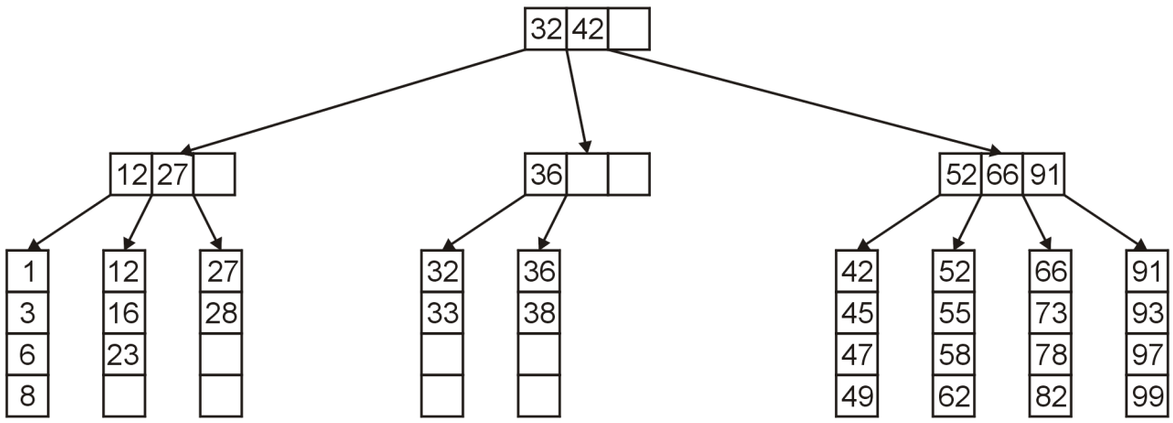


Figure 1a. A B-tree.

a. Insert 17 into the B-tree shown in Figure 1a.

b. Delete 36 from the B-tree shown in Figure 1a.

c. Insert 5 into the B-tree shown in Figure 1a.

2. **Hashing**

2.1 **[2]** Correct this function which should convert a 32-bit integer into an integer between 0 and 999.

```
int h( int n ) {
     return n % 1000;
}
```

2.2 **[2]** Does the multiplicative method for generating a hash value work if the multiplier **c** is a small prime number? Justify your answer referring to the desirable characteristics of a hash value.

For your reference, the following function correctly extracts the 10 bits of **c*n** starting at the 13th bit.

```
const int c = some small prime;

int multiplicative( int n ) {
    return ( (c*n) >> 12 ) & 1023;
}
```

2.3 **[3]** Suppose we are storing numbers in the range 10, ..., 69 in a hash table with seven bins and we are using open addressing with double hashing. The first hash value (initial bin) is the least significant digit modulo 7, and the second hash value (the skip) is the most-significant digit. Insert the integers 32, 13, 52, 23, 43, 47, 34 into this hash table and write your answer in the table provided.

0	1	2	3	4	5	6

2.4 **[2]** Why does our choice of the two hash functions in Question 2.3 not work if the numbers we are storing include all two-digit integers between 00 and 99? Suggest an alterative.

3. **Priority Queues**

3.1 **[4]** Insert the numbers 13, 5, 2, 9, 12, 4, 7, 10 (in that order) into an initially empty min-heap which is stored as a complete tree. You may draw your answer as a tree or as an array. Show the resulting heap after each of your insertions.

3.2 **[2]** Dequeue the minimum from the full min heap implemented as a complete array (as shown in class) and write your answer in the next table. The first row gives the array indices.

0	1	2	3	4	5	6	7	8	9	10	11	12
x	5	7	14	25	19	15	17	32	64	48	21	18

0	1	2	3	4	5	6	7	8	9	10	11	12
x												

3.3 **[4]** A programmer tried to implement a function which inserts an element into a min-heap–as–complete–tree (the version we saw in class). The variable **`count`** stores the number of items currently in the heap. The array was allocated with the statement

```
    array = new Object[array_size];
```

The code is:

```
template <class Object>
void MinHeap<Object>::insert( const Object & obj ) {
    if ( count == array_size ) {
        throw new HeapFullException;
    }

    array[count] = obj;

    for ( int posn = count;
          array[posn/2] < array[posn] && posn >= 0;
          posn = posn/2 ) {
        Object tmp = array[posn];
        array[posn] = array[posn/2];
        array[posn/2] = tmp;
    }

    ++count;
}
```

Correct the logic of this programmer's code.

```
template <class Object>
void MinHeap<Object>::insert( const Object & obj ) {









}
```

4. **Sorting**

4.1 **[2]** Consider sorting a list with n entries. In the worst-case scenario, insertion sort requires $\sum_{i=1}^{n-1} i = \frac{n(n-1)}{2}$ swaps to sort this list while, in the average case, insertion sort requires $\sum_{i=1}^{n-1} \frac{i}{2} = \frac{n(n-1)}{4}$ swaps. In both these cases, the run time is therefore $\mathbf{O}(n^2)$. In the best-case scenario, where the list is already sorted, no swaps required and therefore the run time is $\mathbf{O}(1)$.

Do you agree with this analysis? Why or why not?

You may do question 4.2a or 4.2b, but not both

4.2a **[3]** Perform a heap sort of the numbers 13, 6, 10, 5, 2, 9 in place using a max-heap.

13	6	10	5	2	9

4.2b **[2]** Perform a heap sort of the 13, 6, 10, 5, 2, 9 by placing them into a min-heap and then removing them.

4.3 **[3]** Use the merge sort algorithm to sort the following data. Automatically sort any sub-list of size two. Indicate at each step which operation (partitioning, sorting, or merging) is being performed. Indicate the current partitions with lines.

Operation	8	4	2	5	1	6	5	3

4.4 **[4]** For $n > 1$, both merge sort and quick sort have run times defined by

$$T(n) = 2\,T(n/2) + \Theta(n).$$

For each, explain what is occurring which requires the $\Theta(n)$ run time, and when it occurs.

4.5 **[3]** Use radix sort to sort the numbers

1023, 1020, 3021, 2013, 2130, 1020, 3101, 2013.

5. **Graph Algorithms**

5.1 **[4]** Figure 5a shows an undirected weighted graph.

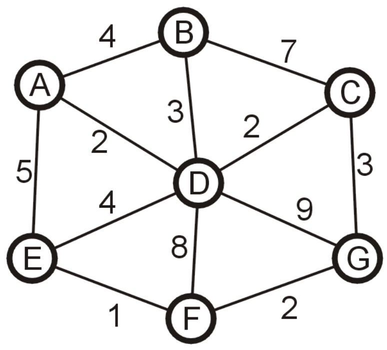
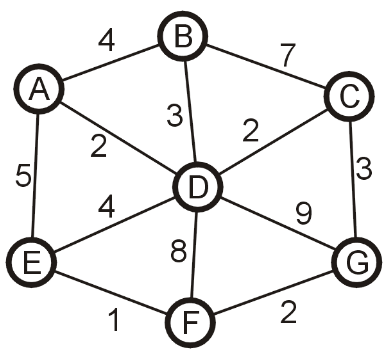
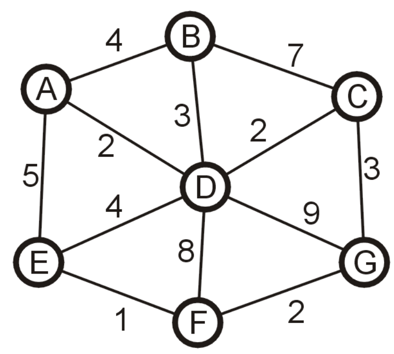
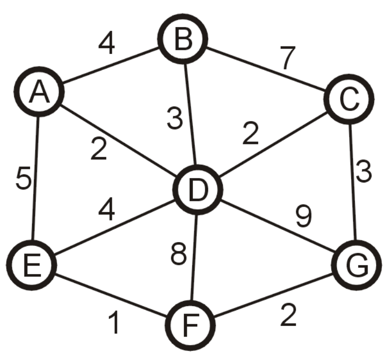


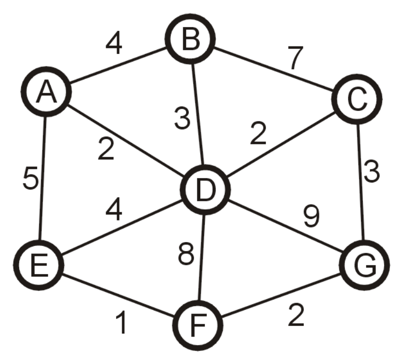
Figure 5a. An undirected graph.

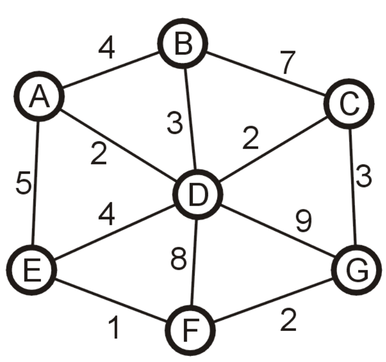
Dijkstra’s algorithm works as well for undirected graphs as it does for directed graphs. Use Dijkstra’s algorithm to find a minimum-length path from node A to node G. Show each step in the algorithm.

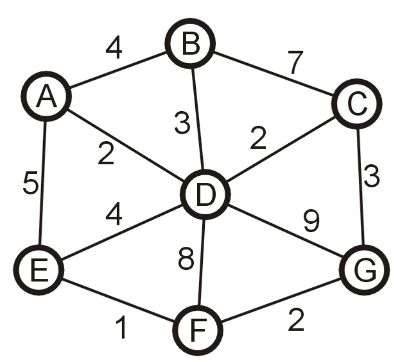


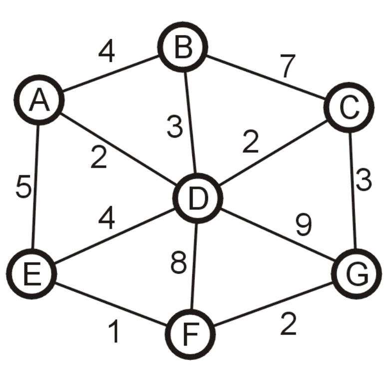


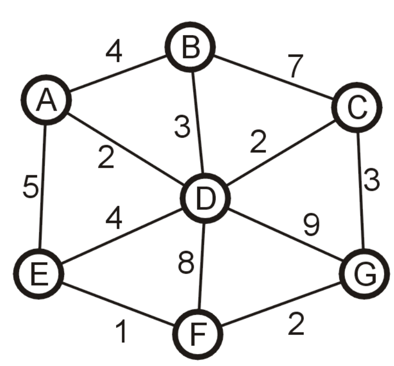




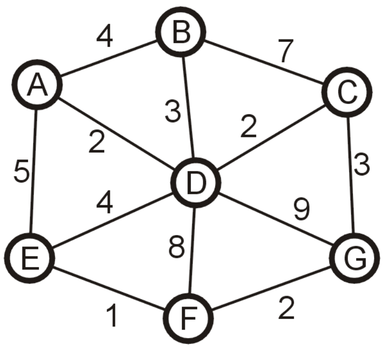


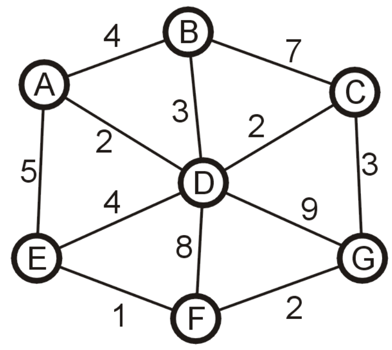


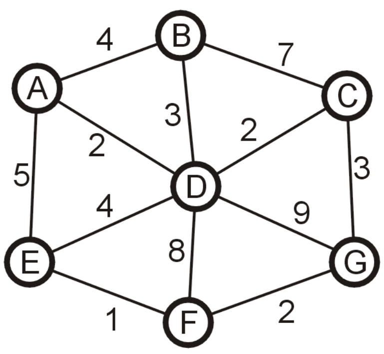


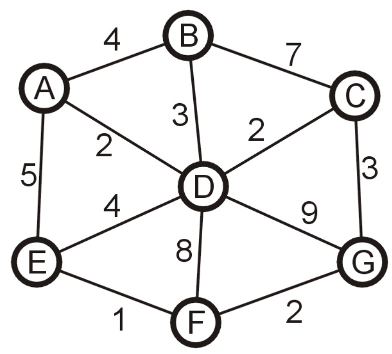


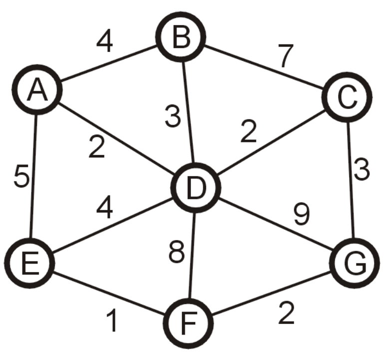
5.2 **[4]** Using Prim's algorithm, find a minimum spanning tree of the graph shown in Figure 5a using node **A** as the root node. You may use tables or the graphs provided below. Show each step and draw the final tree at the bottom of the page.

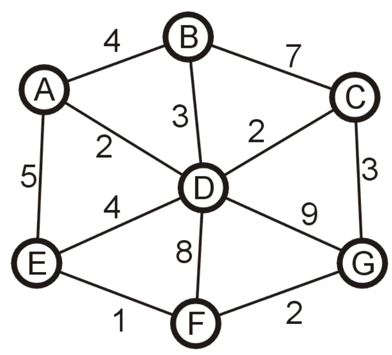


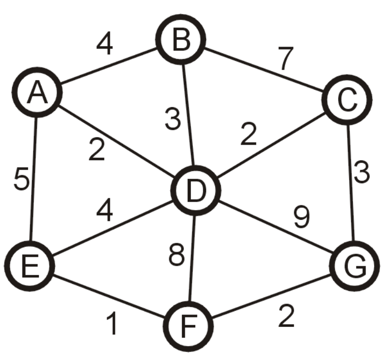


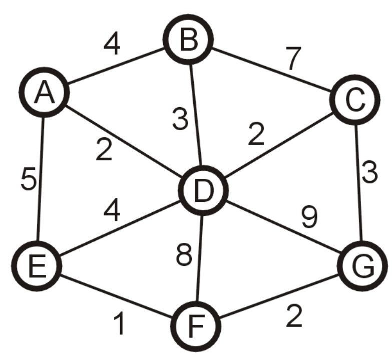












6. **Algorithms**

6.1 **[3]** Consider the (ordered) skip list shown in Figure 6a where the length of the next-pointer array is between 1 and 3. Perform the three given insertions, drawing all pointer connections.

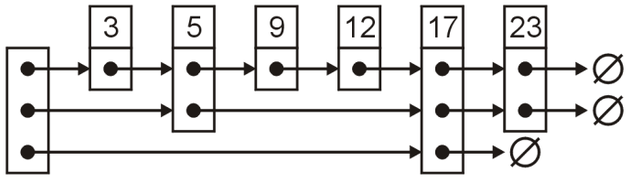
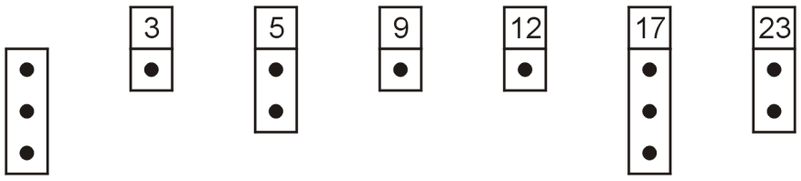
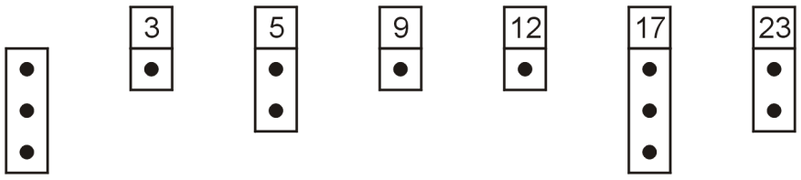


Figure 6a. A skip list.

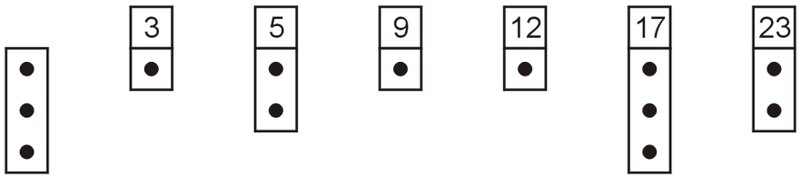
a. Insert 1 into the skip list shown in Figure 6a using a next-pointer array with a length of 2.



b. Insert 10 into the skip list shown in Figure 6a using a next-pointer array with a length of 3.



c. Insert 25 into the skip list shown in Figure 6a using a next-pointer array with a length of 1.



6.2 **[6]** Sudoku™ is a game where you try to arrange numbers in a square given the following two rules:

1. Each number must appear exactly once in each row and column, and
2. Each number must appear once in each sub-square.

Given an incomplete solution, your goal is to try to find a full solution. For example, Figure 6b shows an incomplete solution for a 4 x 4 square with numbers 1, 2, 3, 4 and four sub-squares of size 2 x 2. The unique full solution for this incomplete solution is given on the right.

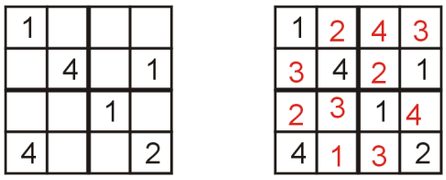


Figure 6b. An incomplete solution and its associated unique full solution.

Devise a backtracking algorithm to finding a full solution for a given incomplete solution. Your pruning rules should be based solely on the two rules given above.

Using the back of the previous page, implement your technique to show that the incomplete solution shown in Figure 6c has no associated full solution.

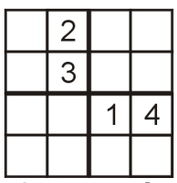
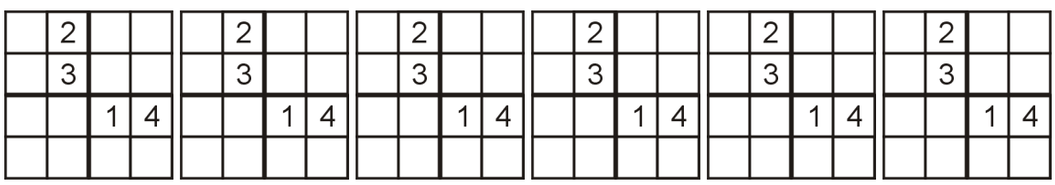


Figure 6c. An incomplete solution.

Some copies of the incomplete solution are provided here for intermediate calculations. You will be awarded marks based on how closely you follow your algorithm (assuming your algorithm is well defined).



7. **Red-Black Trees**

7.1 **[3]** Every node in a red-black tree is given a colour, either red or black. What are the three additional rules which define a red-black tree?

7.2 **[5]** Figure 7a shows a red-black tree with 9 nodes (white background = red). Perform the following three insertions into this tree. Indicate how you are representing red and black nodes in a legend.

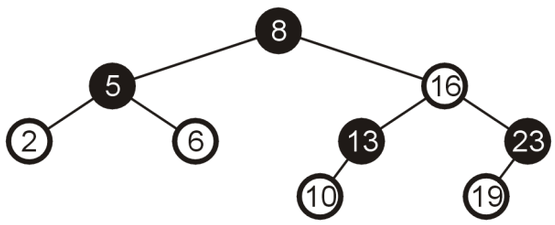


Figure 7a. A red-black tree.

a. Insert 25 into the red-black tree shown in Figure 7a.

b. Insert 7 into the red-black tree shown in Figure 7a.

c. Insert 12 into the red-black tree shown in Figure 7a.

8. **Projects**

8.1 **[6]** In Project 4 (and on the midterm examination) you implemented an **`UndoRedoStack`**. Add the following modification: allow an event to be associated with a boolean value, **`true`** indicating that it is a check point. Instead of just being able to undo a single event, the user has the option of undoing (rolling back) all events up to, but not including, the last event which was identified as a check point. All events which were undone should be returned in stack and should also be placed into the redo stack in the appropriate order to allow the user to redo those events. The returned stack should have the events in the same order as they appear in the redo stack. If no event has been marked as a check point, all events in the undo stack should be undone. You may define your own classes or add additional private class members, member functions, or constructors, whatever you deem necessary. Implement the four functions which are listed on the next two pages.

For your reference, a solution to the **`UndoRedoStack`** is provided on the last page.

```
template <class Object>
class UndoRedoRollbackStack {
      private:
            StackAsList<Object> undo_stack;
            StackAsList<Object> redo_stack;
            // add additional private class members here




      public:
            // default constructor, copy constructor, and destructor

            bool can_undo() const;
            bool can_redo() const;

            void event( const Object &, bool );

            Object undo();
            Object redo();
            StackAsList<Object> rollback();
};

template<class Object>
void UndoRedoRollbackStack<Object>::event( const Object & obj,
                                           bool is_checkpoint ) {









}
```

```
template<class Object>
Object UndoRedoRollbackStack<Object>::undo() {









}

template<class Object>
Object UndoRedoRollbackStack<Object>::redo() {









}

template<class Object>
StackAsList<Object> UndoRedoRollbackStack<Object>::rollback() {









}
```

```
template <class Object>
class UndoRedoStack {
      private:
            StackAsList<Object> undo_stack;
            StackAsList<Object> redo_stack;

      public:
            // default constructor, copy constructor, and destructor

            bool can_undo() const;
            bool can_redo() const;

            void event( const Object & );

            Object undo();
            Object redo();
};

template<class Object>
bool UndoRedoStack<Object>::can_undo() const {
   // returns true if the undo stack is not empty
   return !undo_stack.empty();
}

template<class Object>
bool UndoRedoStack<Object>::can_redo() const {
    // returns true if the redo stack is not emtpy
    return !redo_stack.empty();
}

template<class Object>
void UndoRedoStack<Object>::event( const Object & obj ) {
    // empty the redo stack
    while ( !redo_stack.empty() ) {
       redo_stack.pop();
    }

    // place the object onto the undo stack
    undo_stack.push( obj );
}

template<class Object>
Object UndoRedoStack<Object>::undo() {
   if ( undo_stack.empty() ) {
      throw new UnableToUndoException;
   }

    // grab the top object of the undo stack
    Object obj = undo_stack.top();
    // pop the undo stack
    undo_stack.pop();
    // push the object onto the redo stack
    redo_stack.push( obj );

    return obj;
}

template<class Object>
Object UndoRedoStack<Object>::redo() {
    if ( redo_stack.empty() ) {
       throw new UnableToRedoException;
    }

    // grab the top of the redo stack
    Object obj = redo_stack.top();
    // pop to redo stack
    redo_stack.pop();
    // push the object onto the undo stack
    undo_stack.push( obj );

    return obj;
}
```