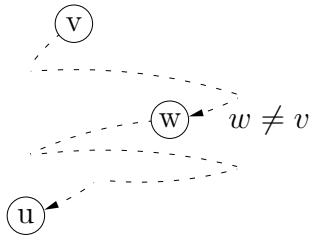
Name: _____

Please submit this homework by following the homework submission instructions on the class Web site. Reminder: You are welcome, and encouraged, to use any resources (e.g., Web sites) to help you with your work. However, *all such help must be clearly noted* in your submissions. Further, no matter what you use, *you must be able to explain* how and why it works. Refer to the class policy for details, and ask for clarifications if you are unsure if something is allowed. Questions marked with $\star$ are optional and may be answered for extra credit.

1. (1 pt.) Write your name in the space provided above.

2. (1 pt.) Package and submit your solutions to the programming questions as in previous assignments. After uploading your jar file to the FTP server, complete the following:
   File name: _____ Size, in bytes: _____

3. (8 pts.) Consider a directed acyclic graph $G = (V, E)$ and a pair of vertices $u, v \in V$ such that u is reachable from v. We use $M(u, v)$ to denote the set of all vertices w such that $w \neq v$, w is reachable from v, and u is reachable from w. Note that, unless $u = v$, $M(u, v)$ contains at least one vertex, u, since $w = u$ is permitted. The relationships between u, v, and w are illustrated below, where the dashed lines represent directed paths.



   Refer to the definition of the graphs B_n in Question 6a of Homework 5. For each vertex v of of the graphs B_1, B_2, and B_3, compute $M(v_\emptyset, v)$, where $v_\emptyset$ denotes the vertex representing the empty set. Write each value of $M(v_\emptyset, v)$ next to the corresponding vertex in your depiction of the graphs B_1, B_2, and B_3.

4. (10 pts.) Consider a directed acyclic graph $G = (V, E)$ and a pair of vertices $u, v \in V$ such that u is reachable from v. We define a function $\mu(u, v)$ recursively as follows, using the definition of $M(u, v)$ from Question 3:

$$\mu(u,v) = \begin{cases} 1 & \text{if } u = v \\ -\sum_{w \in M(u,v)} \mu(u,w) & \text{otherwise} \end{cases}$$

For each vertex v of of the graphs B_1, B_2, and B_3, compute $\mu(v_\emptyset, v)$, where $v_\emptyset$ denotes the vertex representing the empty set. Write each value of $\mu(v_\emptyset, v)$ next to the corresponding vertex in your depiction of the graphs B_1, B_2, and B_3 of Question 3.

5. (20 pts.) For this programming question, and those that follow, use the usual packaging and submission procedure from earlier assignments. Provide a class `MyGraph` that implements the interface `GraphExercises` of Figure 1, as described further in this and the following questions.

   We use the `Vertex`, `Edge`, and `Graph` classes from the textbook (along with related classes) as outlined in Figures 14.6–14.8 of the textbook. You may modify the textbook's versions of the `Vertex`, `Edge`, and `Graph` classes if you wish, as long as the semantics of the existing methods are unchanged. In fact, an efficient implementation of some of the following requires some such modifications. *In your README file, clearly describe the changes, if any, you make to these classes.*

   (a) Write a method `numVertices` that returns the number of vertices in the graph given as argument.

   (b) Write a method `allVertices` that returns a set containing all vertices in the graph given as argument.

   (c) Write a method `numEdges` that returns the number of edges in a graph given as argument.

   (d) Write a method `allEdges` that returns a set containing all edges in the graph given as argument.

6. (20 pts.) Refer to the interface `GraphExercises` of Figure 1.

   (a) Write a method `createBn` that, given a positive integer n, creates the graph B_n of Question 6 of Homework 5, returning the resulting `Graph` object. The `name` of a vertex in the returned graph must be the string obtained by concatenating (in ascending order) the elements of the subset of $[n]$ represented by the vertex. Recall that the textbook's implementation of the `Vertex` class associates a string `name` with each vertex. For example, the name of a vertex that represents the set $\{3, 7, 5\}$ must be the string *357*.

   (b) Write a method `inducedSubgraph` that returns a new graph that is the subgraph of the graph given as first argument induced by the set of vertices given as the second argument. (Recall the definition of an induced subgraph from Question 6g of Homework 5.)

   (c) Write a method `randomVertices` that returns a set of vertices of the graph given as first argument. The number of vertices in the set is provided as the second argument. The returned vertices should be selected uniformly randomly from the vertices of the given graph.

7. (20 pts.) Refer to the interface `GraphExercises` of Figure 1.

   (a) Write a method `reachableVertices` whose arguments are a graph and a vertex in that graph. It returns the set $R(v)$ of vertices reachable from the given vertex

v by traversing zero or more edges (in the forward direction only). Note that the input vertex v is always included in the set $R(v)$.

(b) Write a method `tweenVertices` corresponding to the function M defined in Question 3. That is, `tweenVertices(g,u,v)` returns the set of vertices $M(u, v)$ in the given graph g.

(c) Write a method `mu` corresponding to the function μ defined in Question 4. That is, `mu(g,u,v)` returns the integer $\mu(u, v)$ in the given graph g.

(d) Write a method `nameAsInt` that returns the integer obtained by interpreting a vertex's name (a string as defined in Question 6a) as an integer. For example, if a vertex has name *357* (a string), the result of this method is the integer 357. Note that this function is not one-to-one; for example, both $\{1, 22, 3\}$ and $\{13, 22\}$ are mapped to 1322.

8. (20 pts.) Refer to the interface `GraphExercises` of Figure 1.

(a) Given a graph $G = (V, E)$ and an integer function $f : V \to \mathbb{Z}$ defined on G's vertices, we define the *outward aggregation* of f as the function $g : V \to \mathbb{Z}$ where

$$g(v) = \sum_{x \in R(v)} f(x)$$

and $R(v)$ is the set of vertices reachable from v, as defined in Question 7a. Write a method `outwardAggrOfNameAsInt` that returns $g(v)$ where f is the function `nameAsInt` of Question 7d.

(b) Given a graph $G = (V, E)$ and an integer function $f : V \to \mathbb{Z}$ defined on G's vertices, we define the *outward aggregation with* μ of f as the function $h : V \to \mathbb{Z}$ where

$$h(v) = \sum_{x \in R(v)} f(x)\mu(x, v)$$

and $R(v)$ is the set of vertices reachable from v, as defined in Question 7a. (The definition differs from that in Question 8a only in the $\mu(x, v)$ factor.) Write a method `outwardAggrMuOfNameAsInt` that returns $h(v)$ where f is the function `nameAsInt` of Question 7d.

(c) Write a method `sampleRun` that takes a positive integer as input and prints, to standard output, one line for each vertex in the graph B_n (Question 6 of Homework 5). The line for vertex v should contain the following fields, separated by single tab characters:

(1) a string representation of the set represented by v (see below),
(2) the name of v (Question 6a),
(3) the value of `outwardAggrOfNameAsInt`(v),
(4) the value of `outwardAggrMuOfNameAsInt`(v), and

(5) the value of $\mathrm{hg}(v)$, where the definition of $\mathrm{hg}(v)$ is similar to that of $h(v)$ in Question 8b, but with the function f replaced with the function `outwardAggrOfNameAsInt`.

The string representation of a set of integers $S = \{s_1, s_2, \ldots, s_k\}$ is the string {t_i, t_2, ..., t_k } where the t_i are the s_i sorted in ascending order. For example, the set $\{5, 1, 7, 2\}$ is represented by the string "{1, 2, 5, 7}" (there is a single space after each comma).

9. (5 pts.) ⋆ Refer to the interface `GraphExercises` of Figure 1. Implement a method `sampleRun2` whose output is similar to that of method `sampleRun` of Question 8c, but instead of using the graph B_n, use a random induced subgraph of B_n. In more detail, given parameters n and m, this method should create the graph B_n, produce a random subset S of m vertices of this graph, and the subgraph $B_n|_S$ of B_n induced by S. The method should then print to standard output information similar to that outlined for Question 9, but using the graph $B_n|_S$ instead of B_n.

10. (5 pts.) ⋆ Recall the definition of the outward aggregation of a function from Question 8a. Write a method `outwardAggrOfF` that is similar to `outwardAggrOfNameAsInt` from that question, except that the function f (used by the definition in that question) is not fixed but is provided only when the method is invoked. Determine the appropriate method signature for `outwardAggrOfF` to permit such parametrization of f and implement the method. Hint: Consider functors.[1] Provide a driver program and a few test cases for your method (using various functions f). Include your implementation with the rest of your submission and indicate, in your README file, how this method is tested.

11. (10 pts.) ⋆ What pattern, if any, emerges in the output of Questions 8c and 9? Describe the pattern as precisely as possible. Make a claim based on this pattern. Provide experimental evidence (beyond that from the earlier questions) in support of this claim. Prove your claim formally.

12. (20 pts.) ⋆ For additional extra credit, you may submit your solution to the extra-credit Question 9 of Homework 4. Include all files necessary to compile and test your solution to this question along with the rest of your work for this homework. Be sure to include the relevant details in the README file.

[1] Mark Allen Weiss, *Data Structures and Problem Solving Using Java*, 3rd edition (Addison-Wesley, 2006), Section 4.8.

```
/**
   We use the class Graph (and related classes) from the textbook.
 */
public class GraphExercises {

    /**
       @param g a non-null graph (may be empty).
       @return the number of vertices in g.
     */
    public static int numVertices(Graph g);

    /**
       @param g a non-null graph (may be empty).
       @return a set containing all vertices in g.
     */
    public static Set<Vertex> allVertices(Graph g);

    /**
       @param g a non-null graph (may be empty).
       @return the number of edges in g.
     */
    public static int numEdges(Graph g);

    /**
       @param g a non-null graph (may be empty).
       @return a set containing all edges in g.
     */
    public static Set<Edge> allEdges(Graph g);

    /**
       @param n a positive integer.
       @return A new graph Bn whose vertices are all subsets of the
       set {1, 2, ..., n} and whose edges are vertices (u,v) such that
       v is a maximal proper subset of u (i.e., there is no proper
       subset w of u that is a proper superset of u).
     */
    public static Graph createBn(int n);

    /**
       @param g a non-null graph.
       @param v a set of vertices of g.
       @return a new graph that is the subgraph of g induced by the
       vertices in v.
     */
    public static Graph inducedSubgraph(Graph g, Set<Vertex> v);
```

Figure 1: GraphExercises.java

```
    /**
       @param g a non-null graph.
       @param n a nonnegative integer no greater than the number of
       vertices in g.
       @return a set of n vertices of g, chosen uniformly randomly.
     */
    public static Set<Vertex> randomVertices(Graph g, int n);

    /**
       @param g a non-null graph.
       @param v a vertex in g.
       @return a set containing the vertices in g that are reachable
       from v by traversing zero or more edges (in the forward
       direction only).
     */
    public static Set<Vertex> reachableVertices(Graph g, Vertex v);

    /**
       @param g a non-null graph.
       @param u a vertex in g.
       @param v another vertex in g, may be the same as u.
       @return the set of vertices w, not equal to v, that are
       reachable from v and from which u is reachable.
     */
    public static Set<Vertex> tweenVertices(Graph g, Vertex u, Vertex v);

    /**
       parameters as in tweenVertices.
       See the recursive function mu defined in the text.
     */
    public static int mu(Graph g, Vertex u, Vertex v);

    /**
       @param v a graph vertex.
       @return the integer interpretation of v's name.
     */
    public static int nameAsInt(Vertex v);

    /**
       @param g a non-null graph.
       @param v a vertex in g.
       @return the outward aggregation of the nameAsInt function (as
       defined in the text), evaluated at v.
     */
    public static int outwardAggrOfNameAsInt(Graph g, Vertex v);
```

Figure 1: `GraphExercises.java` (contd.)

```
    /**
95     @param g a non-null graph.
       @param v a vertex in g.
       @return the outward aggregation with mu of the nameAsInt function (as
       defined in the text), evaluated at v.
     */
100  public static int outwardAggrMuOfNameAsInt(Graph g, Vertex v);

    /**
       @param n the number of vertices in the graph Bn
       Exercise some of the above methods for the graph Bn; see text.
105  */
    public static void sampleRun(int n);

    /**
       @param n the number of vertices in the graph Bn
110    @param m the number of vertices in the induced subgraph.
       Exercise some of the above methods for a random induced
       subgraph of the graph Bn; see text.
     */
    public static void sampleRun2(int n, int m);
115 }
```

Figure 1: `GraphExercises.java` (contd.)