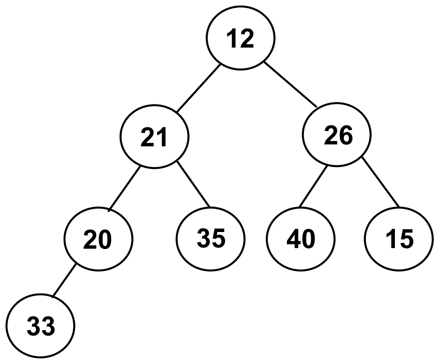
Problem Set 5

Due prior to lecture on Wednesday, December 8
No submissions will be accepted after Sunday, December 12, at 11:59 p.m.
so that we can post solutions before the final exam.

Notes:

- ***Because the grad-credit projects are due soon, there are no extra grad-credit problems for this problem set.***
- ***There are no programming problems for this problem set.***

Written Exercises (100 points total)



1. **Heaps and heapsort** (10 points total)
   a. (3 points) Illustrate the process of turning the tree at right into a max-at-top heap. Show the tree after each sift operation.
   b. (2 points) What is the array representation of the max-at-top heap that you obtain in part a?
   c. (5 points) Heapsort begins by turning the array to be sorted into a heap. Assume that your answer to part b is the result of this process of turning the original array into a heap. Illustrate the remaining steps involved in applying heapsort to this array. Show the contents of the array after each element is put into its final position – i.e., at the end of each iteration of the while loop in the `heapSort` method covered in lecture.

2. **Informed state-space search** (8 points total; 4 points each part)
   You want to use state-space search to solve the following Eight Puzzle:

4	7	5
3		1
6	2	8

   a. What are the first three states to which greedy search would apply the goal test, and what priority would it assign to each of these three states? Give the states in the order in which they would be tested.
   b. What are the first three states to which A* search would apply the goal test, and what priority would it assign to each of these three states? Give the states in the order in which they would be tested.

In answering these questions, you should make the following assumptions:

- Each algorithm follows the approach outlined on the slide entitled "Pseudocode for Finding a Solution" (pg. 8 of the notes on state-space search; pg. 142 of the coursepack). However, previously seen states are *not* discarded.
- When generating the successors of a state, the four operators are applied in the following order: move blank left, move blank right, move blank up, move blank down.
- When adding states to the heap used to store the yet-to-be-considered states, the states are added in the order in which they were generated. For example, the successor generated by moving the blank left would be inserted in the heap first, followed by the successor generated by moving the blank right, etc.
- When determining the priority values, the Manhattan distance heuristic is used.

Please write the states as matrices of numbers, with an underscore for the blank. For example:

```
4 7 5
3 _ 1
6 2 8
```

3. **Tree-based priority queue** (8 points total)
   a. (5 points) A priority queue can be implemented using a binary search tree instead of a heap. Assume that you have an instance *T* of a binary-search-tree implementation (like our LinkedTree class from lecture). Describe how you could use *T* to implement a priority queue, giving pseudocode for both the insert operation (which adds a new item to the priority queue) and the remove operation (which removes the item with the highest priority from the priority queue). You may assume that the search-tree implementation includes *search*, *insert,* and *delete* methods that behave as described in the lecture notes. In addition, you may find it helpful to add an extra method to the binary-search-tree implementation that would be useful in implementing a priority queue. If you do so, you should provide pseudocode for that method as well. Given your proposed implementation, what is the worst-case efficiency of each of the two priority-queue operations (insert and remove)?
   b. (3 points) Your boss suggests that you could improve the efficiency of the *remove* operation in your binary-search-tree-based priority queue by adding a field that stores a reference to the node containing the item with the highest priority. Would this work, or would the resulting *remove* operation still have the same worst-case efficiency that it did before the change? Explain briefly.

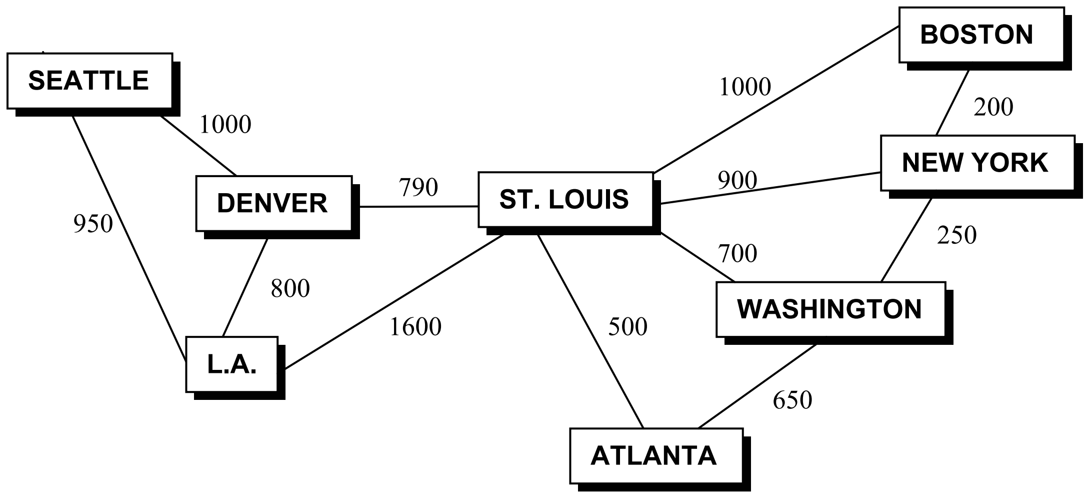
4. **Hash tables** (8 points; 4 points each part)
The following sequence of keys is to be inserted in a hash table of size 8:

   the, my, an, by, do, we, if, to, go

   The hash function assigns to each key the number of characters in the key.

   a. Assume that open addressing with quadratic probing is used to insert the keys. Determine which key causes the table to overflow, and show the table at the point at which it does so.
   b. Now assume instead that open addressing with double hashing is used to insert the keys, with the value of the second hash function being 1 for a noun or pronoun (we, my), 2 for a verb (do, go), 3 for an article (the, an), and 4 for any other part of speech (the rest of the words). Determine which key causes the table to overflow, and show the table at the point at which it does so.

Questions 5-7 refer to the following graph:



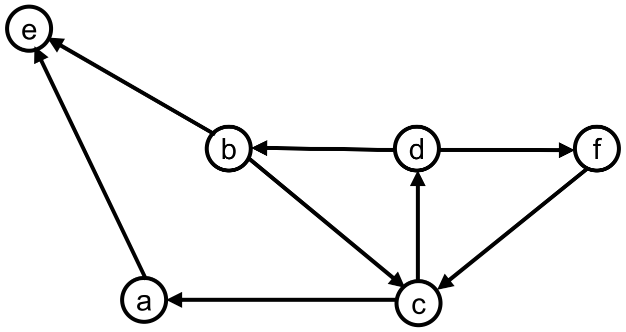
5. **Graph traversals** (8 points; 2 points each part)
Suppose that you have purchased a pass that allows you to fly anywhere you wish along the routes that are shown in the diagram above.
   a. List the order in which you will visit the cities if you start from Atlanta and do a breadth-first traversal. You should assume that the edges of each vertex are stored in order of increasing distance, as we did in the examples in lecture.
   b. What is the path from Atlanta to Boston in the breadth-first spanning tree? Give the path in the form A -> B -> C -> etc., where A, B, and C are vertices.
   c. List the order in which you will visit the cities if you start from Atlanta and do a depth-first traversal. You should assume that the edges of each vertex are stored in order of increasing distance, as we did in the examples in lecture.
   d. What is the path from Atlanta to Boston in the depth-first spanning tree? Give the path in the form A -> B -> C -> etc., where A, B, and C are vertices.

6. **Minimal spanning tree** (6 points)
   A cheaper version of the same pass requires that you confine yourself to flights that are part of a minimal spanning tree for the graph. List the order in which flights/edges will be added to this tree if you build it using Prim's algorithm, starting from Atlanta. Use the form (city1, city2) when specifying an edge.

7. **Dijkstra's shortest-path algorithm** (8 points total)
   Suppose you set out to use Dijkstra's algorithm to determine the shortest distance from Atlanta to every other city in the graph.
   a. (5 points) Make a table showing the order in which the cities are finalized and the minimum distance to each.
   b. (3 points) What path(s) does the algorithm discover from Atlanta to Boston? Include both the final shortest path and any temporary estimates for the shortest path that are later replaced. Give the paths in the form A -> B -> C -> etc., where A, B, and C are vertices.

8. **Determining if two vertices are adjacent** (10 points total)
   a. (3 points) Given the graph representation used in the `Graph` class from lecture, write a simple algorithm for determining if two vertices are adjacent. You may use either pseudocode or Java. Assume that the algorithm takes the two vertices as parameters. You do *not* need to actually implement the algorithm as a method.
   b. (3 points) If a graph with *n* vertices were represented using the `Graph` class from lecture, what would be the worst-case time efficiency of the algorithm from part a? Use big-O notation, and explain your answer briefly.
   c. (4 points) We can make the process of determining if two vertices are adjacent more efficient if we modify the representation of the graph. Describe a modification that will achieve this goal, and explain how the algorithm would be changed to take advantage of the modified representation. State the worst-case time efficiency of the more efficient algorithm using big-O notation, and explain your answer briefly.

9. **Directed graphs and topological sort** (8 points total)
   a. (3 points) The graph below is *not* a directed acyclic graph (a DAG) because it includes at least one cycle. Specify all of the cycles in the graph.
   b. (1 point) The graph below can be transformed into a DAG by reversing the direction of exactly one of its edges. Which one?
   c. (4 points) Use topological sort to find *all* possible topological orderings of the vertices in the DAG obtained in part b.



10. **Choosing a graph representation** (6 points total; 2 points each part)
   In lecture, we looked at two different graph representations: one using an adjacency matrix, and one using an adjacency list. For each of the following cases, state which representation would be better, and justify your choice briefly.
   a. The graph has 10,000 vertices and 40,000,000 edges, and you want to use as little space as possible.
   b. The graph has 10,000 vertices and 20,000 edges, and you want to use as little space as possible.
   c. You need to optimize the time efficiency of determining if two vertices are adjacent, and space efficiency is not important.

11. **Comparing data structures** (6 points)
   A supermarket chain wants you to implement an in-memory database that can be used to access facts about the products they sell. Although a snapshot of this database will be periodically copied to disk, its contents fit in memory, and your component of the application will operate only on data stored in memory.

   Here are the requirements specified by the managers of the supermarket chain:

   - They want to be able to retrieve product records by specifying the name of the product.
   - They want to be able to specify the first n characters of a product name and to retrieve all records that begin with those characters.
   - They want the time required to retrieve a record to be as efficient as possible – on the order of 20 operations per retrieval, given a database of approximately one million records.
   - They want to be able to increase the size of the database – adding large sets of new records – without taking the system offline.

Given this list of requirements, which data structure would be the better choice for this application, a balanced search tree or a hash table – or would these two data structures work equally well? Explain your decision, specifying as many reasons as possible for the choice that you make.

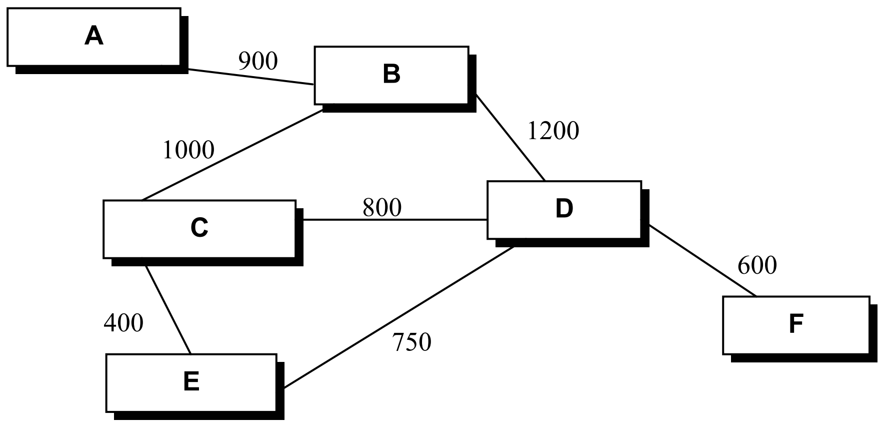
12. **Building bridges** (8 points)
(This problem comes from M. Goodrich and R. Tamassia.) There are eight small islands in a lake, and the state wants to build seven bridges to connect them so that each island can be reached from any other one via one or more bridges. The cost of constructing a bridge is proportional to its length. The distances between pairs of islands are given in the table below. Find which bridges to build to minimize the total construction cost. Your answer should apply one of the graph algorithms that we covered in lecture. **Describe briefly** which algorithm you employed, and explain why it is the appropriate algorithm for this problem.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	-	240	210	340	280	200	345	120
2	240	-	265	175	215	180	185	155
3	210	265	-	260	115	350	435	195
4	340	175	260	-	160	330	295	230
5	280	215	115	160	-	360	400	170
6	200	180	350	330	360	-	175	205
7	345	185	435	295	400	175	-	305
8	120	155	195	230	170	205	305	-

13. **Alternative MST algorithm** (6 points)
Prim's algorithm is just one possible algorithm for finding a minimum spanning tree. Another such algorithm was developed by Kruskal:

```
kruskal_MST(Graph g) {
    put each of g's vertices in its own set
    while (there is more than one set) {
        let e = the minimum-cost edge that hasn't been considered
        if (e connects vertices that are in different sets) {
            add e to the MST
            merge the sets containing the vertices connected by e
        }
    }
}
```

Note that this algorithm considers the edges in order of increasing cost. In addition, at an intermediate stage of the algorithm, there may be multiple trees that are not connected to each other, although they will ultimately be joined together to form a single MST.



For example, if we applied Kruskal's algorithm to the graph above, we would start out with the following sets: { A }, { B }, { C }, { D }, { E }, { F }

We would consider the edge (C, E) first, because it has the lowest cost (400). Because it connects vertices in different sets, we would add this edge to the tree and merge the sets involved to get: { A }, { B }, { C, E }, { D }, { F }

We would next consider the edge (D, F), because it has the smallest remaining cost. Because it connects vertices in different sets, we would add this edge to the tree and merge the sets involved to get: { A }, { B }, { C, E }, { D, F }

We would next consider the edge (D, E), because it has the smallest remaining cost. Because it connects vertices in different sets, we would add this edge to the tree and merge the sets involved to get: { A }, { B }, { C, D, E, F }

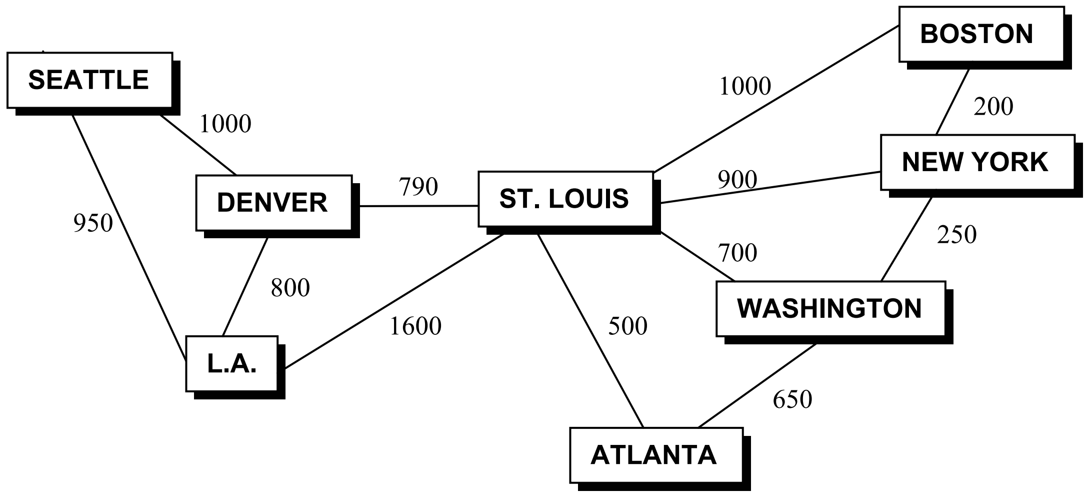
We would next consider the edge (C, D), because it has the smallest remaining cost. Because it connects vertices that are *already in the same set*, we would *not* add this edge to the tree.

We would next consider the edge (A, B), because it has the smallest remaining cost. Because it connects vertices in different sets, we would add this edge to the tree and merge the sets involved to get: { A, B }, { C, D, E, F }

We would next consider the edge (B, C), because it has the smallest remaining cost. Because it connects vertices in different sets, we would add this edge to the tree and merge the sets involved to get: { A, B, C, D, E, F }

Finally, we would consider the edge (B, D). Because it connects vertices in the same set, we would *not* add this edge to the tree.

Apply Kruskal's algorithm to the airline graph from earlier in the problem set, which is reproduced below. List the edges of the MST in the order in which the algorithm would add them, using the format (city1, city2) for an edge.



Submitting Your Work

First, you need to obtain the Makefile for this assignment. To do so, you should log into `nice.harvard.edu` and enter the following command: `gethw 5`

This will create a directory named `ps5` in your Harvard home directory containing the Makefile that you need for this assignment. To change into this directory, the command is: `cd ~/ps5`

Once you are ready to submit, make sure that the file containing your work is in your `ps5` directory on `nice.harvard.edu`.

To submit your assignment, enter the following command: `make submit`

To check which files were submitted, enter the following command: `make check`

Important note: the submit commands only work if you are logged onto *nice*`.harvard.edu`.