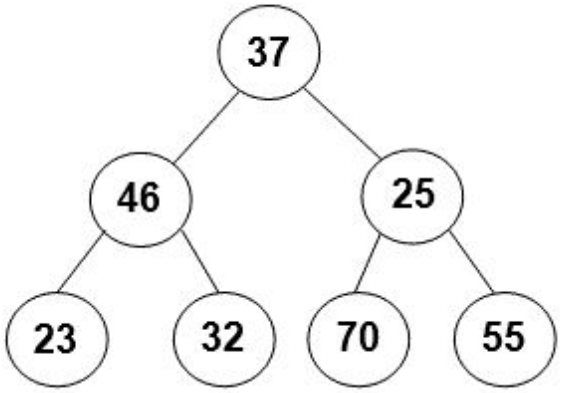
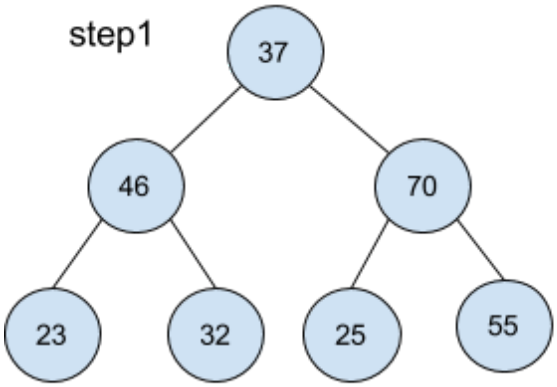
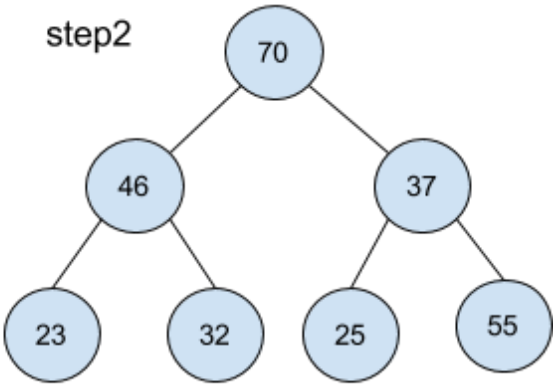
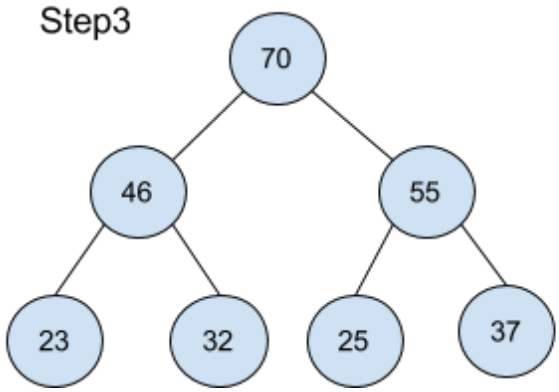
Name: Yiling Jiang
Email: yiling124@gmail.com
Problem Set 5

Problem 1:









2. What is the array representation of the max-at-top heap that you obtain in part 1?
[70, 46, 55, 23, 32, 25, 37]

3, Heapsort begins by turning the array to be sorted into a heap. Assume that your answer to part 2 is the result of this process of turning the original array into a heap. Illustrate the remaining steps in using heapsort on 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 from lecture.

[55, 46, 37, 23, 32, 25, 70]
[46, 25, 37, 23, 32, 55, 70]
[37, 25, 32, 23, 46, 55, 70]
[32, 25, 23, 37, 46, 55, 70]
[25, 23, 32, 37, 46, 55, 70]
[23, 25, 32, 37, 46, 55, 70]

Problem 2: Hash tables

`1, cat, goat, dog, bird, bison, ant, flea, bat, duck`

1, Assume that *linear probing* is used to insert the keys. Determine which key causes overflow, and show the table at that point.

0	“ant”
1	“flea”
2	“bat”
3	“cat”
4	“goat”
5	“dog”
6	“bird”
7	“bison”

When we need to add “duck” to the hash table , the key = 4, cause overflow

2, Now assume that *quadratic probing* is used. Determine which key causes overflow, and show the table at that point.

0	“ant”
1	“flea”
2	“bat”
3	“cat”
4	“goat”
5	“bird”
6	“bison”
7	“dog”

When we need to add “duck” to the hash table , the key = 4, cause overflow. As key = 4 is already occupied, we then tried 4 + 1^2 and 4 + 2^2, which are both occupied. Besides, there is no more empty spot for “duck”. If I insisting add it to the hashtable, it will cause overflow.

3, Finally, assume that double hashing is used, with the value of the second hash function based on the first character in the key: a = 1, b = 2, c = 3, d = 4, e = 5, f = 6, g = 7, etc. Determine which key causes the table to overflow, and show the table at the point at which it does so.

0	“bison”
1	“flea”
2	“bat”
3	“cat”
4	“goat”
5	“ant”
6	“bird”
7	“dog”

When we need to add “duck” to the hash table , the key = (h1 = 4 , h2 = 4), the spot is already occupied in the hashtable, and there is no more empty spot for this entry. If we insisting in adding it to the hash table, it will cause overflow

Problem 3: Informed state-space search

1, What are the priorities that greedy search would assign to the following states: state a, state c, state f, and state m?

state a : priority = -16
state c : priority = -17
state f : priority = -18
state m: priority = -17

2. What are the priorities that A* search would assign to the same four states? Assume again that the Manhattan distance heuristic from lecture is used to estimate the remaining cost.

state a : priority = - (16 + 0) = -16
state c : priority = - (17 + 1) = -18
state f : priority = - (18 + 2) = -20
state m: priority = -(17+ 3) = -20

Problem 4: List-based priority queue

1, A priority queue can be implemented using a list instead of a heap. Describe how you could use a list to implement a priority queue in which the remove operation would have a time complexity of O(1).

Using a list to keep track of the top priority, I would make sure that the sequences of the elements are arranged by it's priority, starting with the element with top priority and the last element has the least priority. All the elements are ordered by it's priority with decreasing order.

2, What would be the efficiency of the insert operation in this list-based implementation? Explain your answer briefly.

To insert one element with certain priority, we need to loop through the list to find its position in the list where its previous element has higher priority and its next element has lower priority. The time complexity could be O(n) at worst case scenario, and O(1) at best case scenario where it has the top priority.

Problem 5: Testing for a path from one vertex to another

1,

```
private static boolean existsPathVertices(Vertex start, Vertex end) {
    /* Implement this method for PS 5. */
  if (start == end) return true;
  If (start.done) return false;
  start.done = true;

  Edge e = start.edges;
```

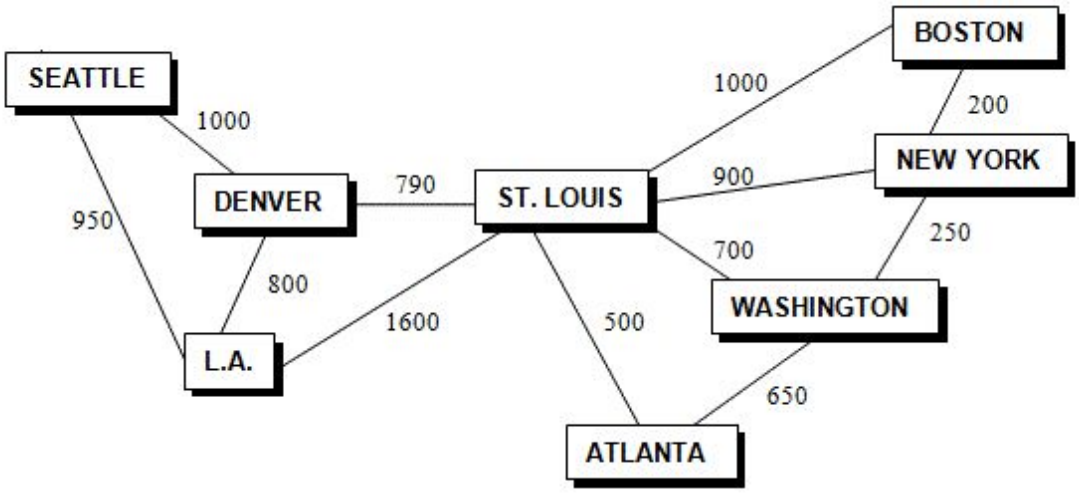
```
  while (e!= null) {
    if (existsPathVerticies(e.end, end)) return true;
  }
  return false;
}
```

2. For a graph with V vertices and E edges, what would be the best-case time efficiency of your method from part 1? What would be the worst-case time efficiency? Use big-O notation, and explain your answers briefly.

In worst case scenario, where all vertices connects to the start vertice, it needs to go through all the vertices to tell if there is one path between start and end vertices. In this case, it will take O(n).

In the base case scenario, it will take O(1) to find the path, if the start vertex and end vertex are connected by one edge.

Problem 6: Graph traversals



1, 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 lecture examples.

Atlanta, st.Louis, Washington, Denver, New York, Boston, L.A. , Seattle

2, What is the path from Atlanta to Boston in the breadth-first spanning tree? Give the path in the form

Atlanta -> St.Louis -> Boston

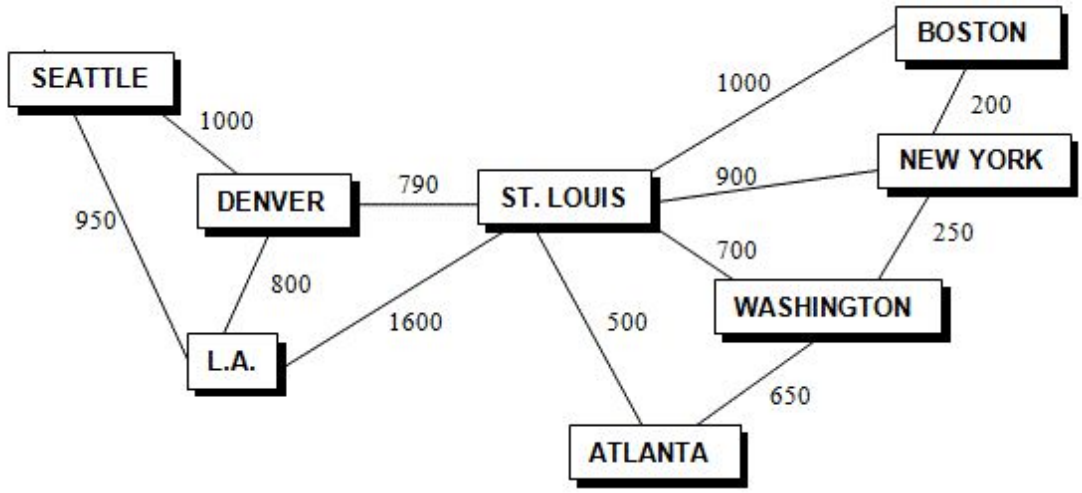
3, List the order in which you will visit the cities if you start from Atlanta and do a depth-first traversal. You should assume that the edges of each vertex are stored in order of increasing distance, as we did in the lecture examples.

Atlanta, ST. Louis, Washington, New York, Boston, Denver, L.A., Seattle

4, What is the path from Atlanta to Boston in the depth-first spanning tree? Give the path in the same form specified above.

Atlanta -> ST.Louis -> Washington -> New York -> Boston

Problem 7: Minimal spanning tree

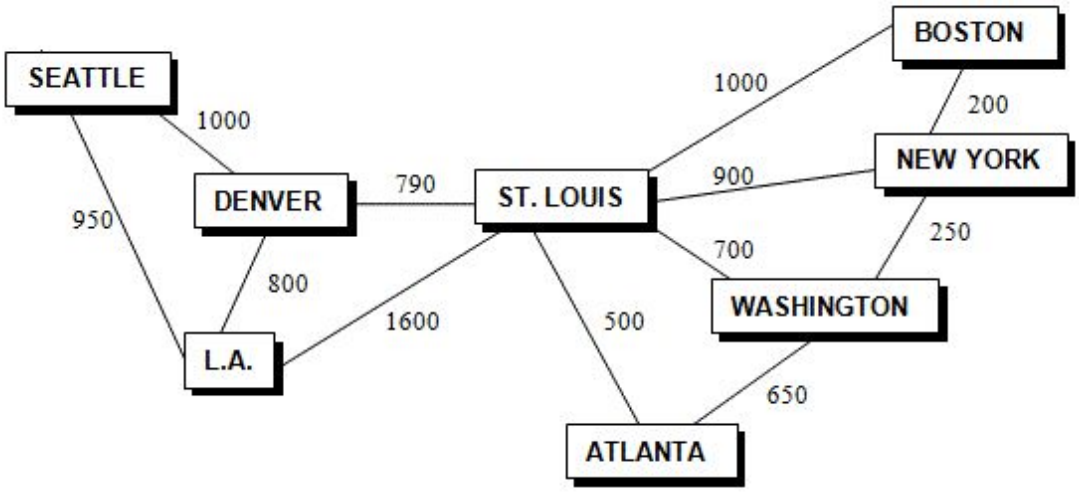


** note: Atl: Atlanta, ST.L: ST Louis, Washtn: Washington, NY: New York, Bostn: Boston, Denv: Denver, Sttl: Seattle

Edge added	Set A	Set B
	{Atl}	{Bost, Denv, NY, L.A, Washtn, ST.L, Sttl}
(Atlanta, ST.L)	{Atl, ST.L}	{Bost, Denv, NY, L.A, Washtn, Sttl}
(Atlanta, Washtn)	{Atl, ST.L, Washtn}	{Bost, Denv, NY, L.A, Sttl}
(Washtn, NY)	{Atl, ST.L, Washtn, NY}	{Bost, Denv, L.A, Sttl}
(NY, Bostn)	{Atl, ST.L, Washtn, NY, Bostn}	{Denv, L.A, Sttl}
(ST.L, Dven)	{Atl, ST.L, Washtn, NY, Bostn, Denv}	{L.A, Sttl}

(Denv, L.A.) | {Atl, ST.L, Washtn, NY, Bostn, Denv, L.A.} | {Sttl}
(L.A., Seattle) | {Atl, ST.L, Washtn, NY, Bostn, Denv, L.A., Sttl} | {}

Problem 8: Dijkstra's shortest-path algorithm

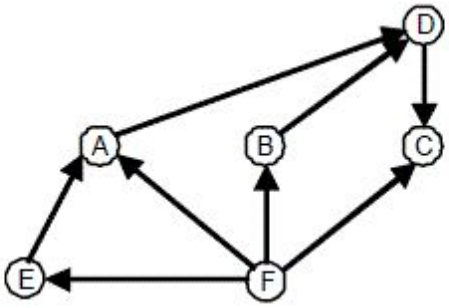


City	Min Distance	Sequence for finalization
Atlanta	0	1
ST Louis	500	2
Washington	650	3
New York	900	4
Boston	1100	5
Denver	1290	6
L.A.	2090	7
Seattle	2290	8

2. 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.

Atlanta -> Washington -> New York -> Boston
Atlanta ->ST.Louis -> Boston

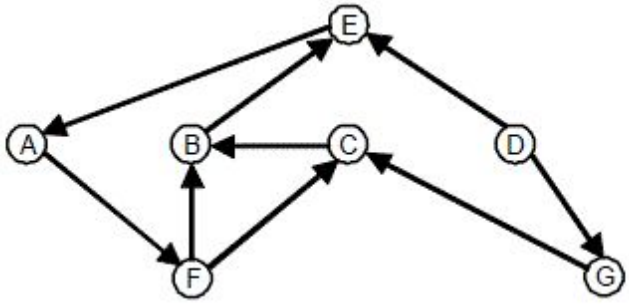
Problem 9: Directed graphs and topological sort



1,
This is a DAG(directed acyclic graph)

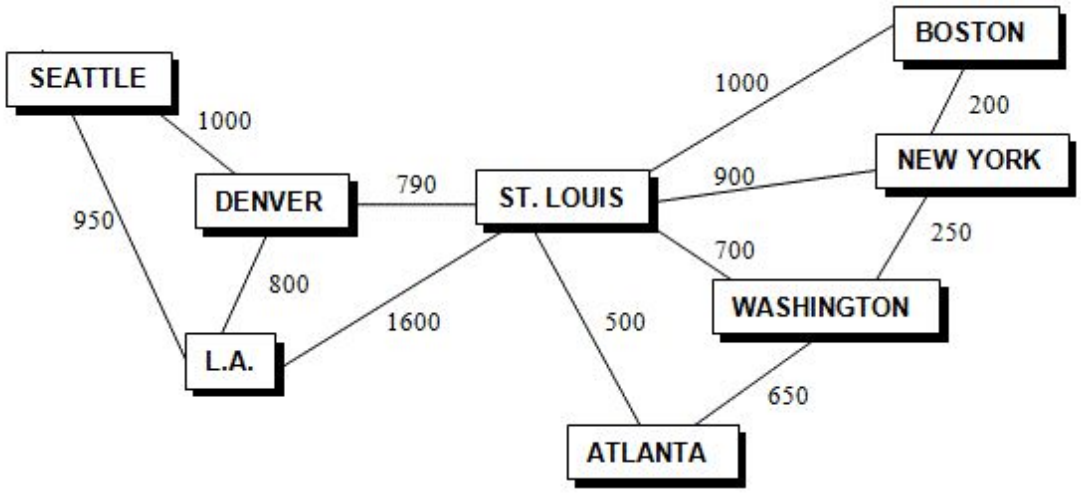
Push	Stack contents (top to bottom)
C	C
D	C, D
B	C, D, B
A	C, D, B, A
E	C, D, B, A, E
F	C, D, B, A, E, F

2, Repeat the process outlined above on the graph below.



This is a not a DAG(directed acyclic graph).
The cycle is : F->B->E->A->F

Problem 10: Alternative MST algorithm



(Boston, New York)
(New York, Washington)
(Atlanta, ST.Louis)
(Washington, Atlanta)
(ST Louis, Denver)
(Denver, L.A.)
(L.A., Seattle)

Problem 11: Maximum-cost spanning tree

```
Prim_MaxST(Vertex start) {
   Put Vertex start in Set A
   Set start.done = true;
   Put all other Vertices in Set B
   Set all vertices in Set B as .done = false;

    while (B is not empty) {
       let e = the maximum-cost edge (Va, Vb)
       add(Va, Vb) to the spanning tree
       Move Vertex Vb from Set B to Set A
     }
}
```

Problem 12: Routing packets

Dijkstra's shortest-path algorithm should be used to solve this problem, as all the servers can be considered as vertices and the cable connections can be considered as edges. The minimum cable length problem can be translated to finding the shortest path(cable) among different vertices(servers). So Dijkstra algorithm should be a good way to solve this problem.

Servers										
Server 1	inf	325	325	325	275					
Server 2	inf	150	150							
Server 3	inf	120								
Server 4	inf	275	275	260						
Server 5	0									
Server 6	inf	inf	inf	inf	775	775	775			
Server 7	inf	inf	inf	inf	inf	1045	1045	890		
Server 8	inf	625	625	625	625	545				

	Next step
Server 1	2
Server 2	2
Server 3	3
Server 4	3
Server 5	
Server 6	2
Server 7	2
Server 8	3