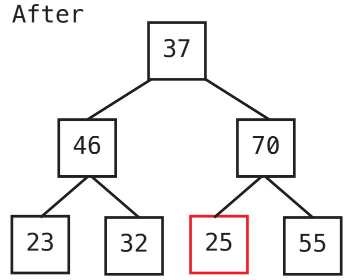
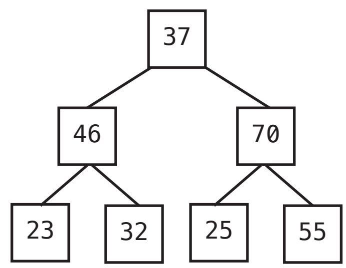
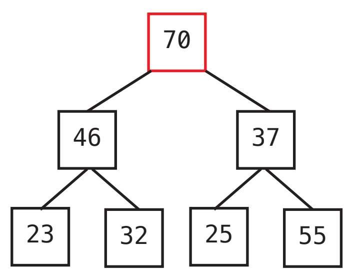
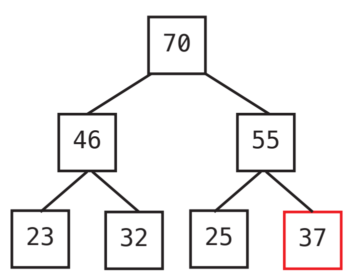
Problem Set 5
December 7, 2011
Graham Schmidt
schmidtg@gmail.com
Harvard ID: 30825489

1. Heap and heapsort

a) Original Contents

37	46	25	23	32	70	55

Operation	After
Sift down contents[2]	
Sift down contents[1]	
Sift down contents[0]	
Continue Sifting down	

b) Representation of array from part a

70	46	55	23	32	25	37

1c)

    Step 1. Remove the largest [70]

```
            55
          /    \
        46      37      >> Sift down 55
       /  \    /
      23   32 25

      array[55 | 46 | 37 | 23 | 32 | 25 | 70]
```

    Step 2. Remove 55 and Sift 25

```
            55              25       (sift 25)      46              46
          /    \          /    \                  /    \          /    \
        46      37      46      37              25      37      32      37
       /  \    /       /  \                    /  \            /  \
      23   32 25      23   32                23    32        23    25

    array[46 | 32 | 37 | 23 | 25 | 55 | 70]
```

    Step 3. Remove 46 and sift down 25.

```
            25              37
          /    \          /    \
        32      37      32      25
       /               /
      23              23

    array[37 | 32 | 25 | 23 | 46 | 55 | 70]
```

    Step 4. Remove 37 and sift down 23.

```
            23              32
          /    \          /    \
        32      25      23      25

    array[32 | 23 | 25 | 37 | 46 | 55 | 70]
```

    Step 5. Remove 32 and sift down 25.

```
            25             25
          /               /
        23              23

    array[25 | 23 | 32 | 37 | 46 | 55 | 70]
```

    Step 6. Remove 25 and sift 23.

```
           23             23

    array[23 | 25 | 32 | 37 | 46 | 55 | 70]
```

2. Hash Tables

```
a) [ flea  ]
   [ bat   ]
   [ cat   ]
   [ goat  ]
   [ dog   ]
   [ bird  ]
   [ bison ]
   [ ant   ]

   'duck' causes overflow.

b) [       ]
```

```
   [       ]
   [ cat   ]
   [ goat  ]
   [ bird  ]
   [ bison ]
   [ dog   ]
   [       ]
```

'ant' causes overflow.

```
c) [ bat   ]
   [ flea  ]
   [ cat   ]
   [ goat  ]
   [ bison ]
   [ bird  ]
   [ dog   ]
   [ ant   ]
```

'duck' causes overflow.

3. Informed state-space search

a) 16

b) The first four state labels for the 'greedy search' algorithm are i, j, k, e with priorities of -13, -13, -14, -14 respectively.

c) For A* the first four states are b, e, i, j with priorities of -16, -16, -16, -16 respectively

4. List-based priority queue

a) To implement a priority queue using a List, we would need to ensure that items are kept in reverse order when inserted into the ADT. The item with the highest assigned priority would be kept at the 'front' of the queue, so an operation like pop () would simply return the first item. We could implement this list as either an array or linked-list. Popping the first item would ensure a time complexity of O(1).

b) The efficiency of the insert operation would be O(n). Since the list would be sorted already, one could implement a modified sort algorithm of say 'insertion sort' that maintains the order in the queue on each insert.

5. Testing for a path between vertices.

a)

```
/**
 * Check if a Path Exists between two vertices.
 * We want to do a depth-first traversal of the tree from the initial vertex
 * and if we come across the vertex, v2, then a path exists
 */
private static boolean pathExists(Vertex v1, Vertex parent, Vertex v2) {

        // Mark this vertex as being visited
        v1.done = true;
        v1.parent = parent;
        Edge e = v1.edges;

        // Check all edges of the existing node
        while (e != null) {
                Vertex w = e.end;

                if (!w.done) {
                        // Found a match
                        if (w.next == v2) {
                                return true;
            }
                        pathExists(w, v1, v2);
        }

                e = e.next;
    }

    return false;
```

}

b) Best-case: O(1) - if the vertices are directly opposite each other for the first edge checked
Worst-case: O(n) - Since we would effectively need to check all the remaining vertices, n - 1

6. Graph Traversals.

a) Breadth-first traversal: Denver, Seattle, O'Hare, San Jose, Atlanata, Washington, New York, Boston, L.A.

b) Denver - O'Hare - Boston

c) Depth-first traversal: Denver, Seattle, O'Hare, Atlanta, Washington, New York, Boston, L.A., San Jose

d) Denver - O'Hare - Atlanta - Washington - New York - Boston

7. Minimal spanning tree

Edge added	Set A	Set B
	{DEN}	{SEA}, {SJ}, {LA}, {OH}, {ATL}, {WAS}, {NY}, {BOS}
(Den, Sea)	{DEN}, {SEA}	{SJ}, {LA}, {OH}, {ATL}, {WAS}, {NY}, {BOS}
(Den, OH)	{DEN}, {SEA}, {OH}	{SJ}, {LA}, {ATL}, {WAS}, {NY}, {BOS}
(OH, ATL)	{DEN}, {SEA}, {OH}, {ATL}	{SJ}, {LA}, {WAS}, {NY}, {BOS}
(ATL, WAS)	{DEN}, {SEA}, {OH}, {ATL}, {WAS}	{SJ}, {LA}, {NY}, {BOS}
(WAS, NY)	{DEN}, {SEA}, {OH}, {ATL}, {WAS}, {NY}	{SJ}, {LA}, {BOS}
(NY, BOS)	{DEN}, {SEA}, {OH}, {ATL}, {WAS}, {NY}, {BOS}	{SJ}, {LA}
(DEN, SJ)	{DEN}, {SEA}, {OH}, {ATL}, {WAS}, {NY}, {BOS}, {SJ}	{LA}
(SJ, LA)	{DEN}, {SEA}, {OH}, {ATL}, {WAS}, {NY}, {BOS}, {SJ}, {LA}	{}

8. Dijkstra's shortest-path algorithm

a) Denver to every other city

6. Atlanta	inf	inf	inf	1800	1800	1800	1800	1800	1800
9. Boston	inf	inf	inf	2000	2000	2000	2000	2000	2000
1. Denver	0	0	0	0	0	0	0	0	0
3. O'Hare	inf	1100	1100	1100	1100	1100	1100	1100	1100
8. New York	inf	inf	inf	1900	1900	1900	1900	1900	1900
5. L.A.	inf	inf	inf	3100	1600	1600	1600	1600	1600
4. San Jose	inf	1200	1200	1200	1200	1200	1200	1200	1200
2. Seattle	inf	900	900	900	900	900	900	900	900
7. Washington	inf	inf	inf	1850	1850	1850	1850	1850	1850

b) Denver to LA?

The algorithm discovers the path: Denver - O'Hare - L.A. as 3100 miles.
Then it discovers a shorter one: Denver - San Jose - L.A. as 1600 miles.

9. Directed graphs and topological sort.

a) Yes, Graph 9-1 is a DAG. One of the possible topological orderings: F, E, A, B, D, C

Push	Stack
C	C
D	D, C
B	B, D, C
A	A, B, D, C
E	E, A, B, D, C
F	F, E, A, B, D, C

b) No, the graph 9-2 is not a DAG.

All the cycles include:
A, F, B, E, A
A, F, C, B, E, A
B, E, A, F, B
B, E, A, F, C, B
C, B, E, A, F, C
E, A, F, B, E
E, A, F, C, B, E
F, B, E, A, F

F, C, B, E, A, F

10. Alternative MST algorithm (Using Kruskal's algorithm)

()	{}	{BOS}, {NY}, {WAS}, {ATL}, {OH}, {DEN}, {SEA}, {SJ}, {LA}
(BOS, NY)	{BOS, NY}	{WAS}, {ATL}, {OH}, {DEN}, {SEA}, {SJ}, {LA}
(NY, WAS)	{BOS, NY, WAS}	{ATL}, {OH}, {DEN}, {SEA}, {SJ}, {LA}
(SJ, LA)	{BOS, NY, WAS} {SJ, LA}	{ATL}, {OH}, {DEN}, {SEA}
(WAS, ATL)	{BOS, NY, WAS, ATL} {SJ, LA}	{OH}, {DEN}, {SEA}
(ATL, OH)	{BOS, NY, WAS, ATL, OH} {SJ, LA}	{DEN}, {SEA}
(DEN, SEA)	{BOS, NY, WAS, ATL, OH} {SJ, LA} {DEN, SEA}	
(DEN, OH)	{BOS, NY, WAS, ATL, OH, DEN, SEA} {SJ, LA}	
(DEN, SJ)	{BOS, NY, WAS, ATL, OH, DEN, SEA, SJ, LA}	

11. Maximum-cost spanning tree

Prim's Maximum Pseudocode

Vertex v is in set A if v.don == true, else it's in set B

Scan to find the next edge to add:
- iterate over the list of vertices on the graph
- for each vertex in A, iterate over its adjacency list
- keep track of the maximum-cost edge connecting a vertex on A to a vertex in B

12. Routing packets

Steps: 1) Build tree from adj. matrix. 2) Start with server 5 and apply Dijkstra's Shortest Path algorithm. 3) Determine which servers to route to from server 5.

Edge	Server	1	2	3	4	5	6	7	8
5-2-1	1	inf	325	325	275	275	275	275	275
5-2	2	inf	150	150	150	150	150	150	150
5-3	3	inf	120	120	120	120	120	120	120
5-3-4	4	inf	275	275	260	260	260	260	260
	5	0	0	0	0	0	0	0	0
5-2-1-6	6	inf	inf	inf	inf	inf	775	775	775
5-2-1-6-7	7	inf	inf	inf	inf	inf	inf	945	890
5-3-4-8	8	inf	625	625	625	545	545	545	545

Dijkstra's is appropriate because it takes the starting point, server #5, and find's the minimum-cost path to the next vertex that hasn't been finalized, thus determining for each vertex, the shortest path to that server from server #5. For example, the path from server 7 to 5 was 945, but after Server 6 was finalized, the shortest known path from 7 to 5 was a distance of 890 (Servers 5-2-1-6-7).