CS 400/600 Homework #4

Shumin Guo

Due Date: Nov. 12th, 2010, beginning of the class.

Note: The first several questions are from the textbook.

1. (10 points): Let G be a simple connected undirected graph (simple graph: no self-edges) with n vertices and m edges. Explain why $O(log_2(m))$ is $O(log_2(n))$.
   ANSWER:
   For a simple connected undirected graph with n vertices, the smallest number of edges is $m = n - 1$, and the largest number of edges is $m = \frac{n(n-1)}{2}$. So we have the following inequation:
   $n - 1 \leq m \leq \frac{n(n-1)}{2}$
   $\Rightarrow log_2(n-1) \leq log_2(m) \leq log_2(n) + log_2(n-1) - 1$
   $\Rightarrow log_2(n-1) < log_2(n) \leq log_2(m) \leq log_2(n) + log_2(n-1) - 1 \leq 2log_2(n)$
   $\Rightarrow log_2(n) \leq log_2(m) \leq 2log_2(n)$
   $\Rightarrow O(log_2(m)) = O(log_2(n))$.

2. (10 points): Let G be a graph whose vertices are the integers 1 through 8, and let the adjacent vertices of each vertex be given by Table 1.

Vertex	adjacent vertices
1	(2,3,4)
2	(1,3,4)
3	(1,2,4)
4	(1,2,3,6)
5	(6,7,8)
6	(4,5,7)
7	(5,6,8)
8	(5,7)

Table 1: Graph adjacent vertices

   Assume that, in a traversal of G, the adjacent vertices of a given vertex are returned in the same order as they are listed in the above table.

   a. Draw G. (See Figure 1.)

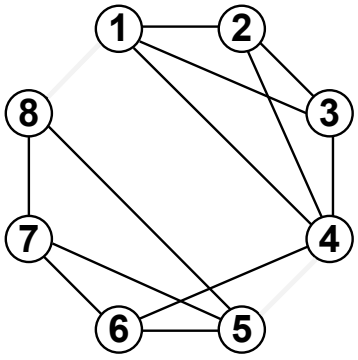

Figure 1: Graph for G.

   b. Given the sequence of vertices of G visited using a DFS traversal starting at vertex 1. $\boxed{1,2,3,4,6,5,7,8}$

   c. Give the sequence of vertices visited using a BFS traversal starting at vertex 1. $\boxed{1,2,3,4,6,5,7,8}$

3. (10 points): Would you use the adjacency list structure or the adjacent matrix structure in each of the following cases? Justify your choice.

a. The graph has 10,000 vertices and 20,000 edges, and it is important to use as little space as possible.
ANSWER:
As $n^2 = 10000^2 = 10^8 \gg 20,000$, there will be a large percentage of space wasted for storage 0. So in this case, adjacency list structure is a better choice.

b. The graph has 10,000 vertices and 20,000,000 edges, and it is important to use as little space as possible.
ANSWER:
As $n^2 = 10000^2 = 10^8$, which is of the same order as $20,000,000$. But if we store graph using adjacency matrix, still 80% of the matrix element is 0. This reason together with the space constraint requirement make the adjacency list a preferable choice over adjacency matrix.

c. You need to answer the query areAdjacent() as fast as possible, no matter how much space you use.
ANSWER:
The time complexity of **areAdjacentVertices()** operation for adjacency list $O(\min(\deg(u),\deg(v)))$ as compared with $O(1)$ for adjacency matrix makes adjacency matrix the better choice.

4. (10 points). Given the following directed, weighted graph, show the values of the estimated distance matrix, D, at each step of Dijkstra′s algorithm as it searches for the minimal-weight path from A to all other vertices, See result at Table 2.

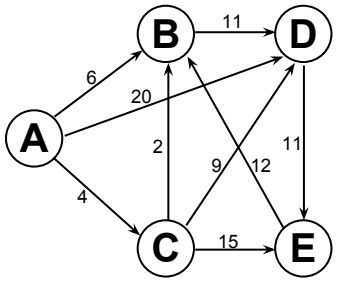

Figure 2: Directed Weighted Graph For Question 4.

	A	B	C	D	E
Initial	0	∞	∞	∞	∞
A	0	6	4	20	∞
B	0	6	4	17	∞
C	0	6	4	13	19
D	0	6	4	13	19
E	0	6	4	13	19

Table 2: Steps for Dijkstra′s algorithm.

5. (20 points). You are given the following weighted graph. (See Figure 3).

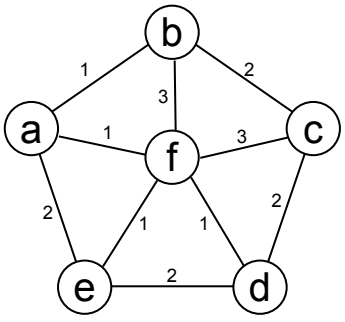

Figure 3: Weighted Undirected Graph.

The adjacency list is:
a $->$ b f e
b $->$ a c f
c $->$ d f b
d $->$ f c e
e $->$ a f d
f $->$ e d c b a

Please use *Prim's* and *Kruskal's* algorithms, respectively, to find the minimum spanning tree, and draw the MST at each step. For the Prim's algorithm, you can assume start with the vertex a.

Please see Figure 4 for Prim's Algorithm and Figure 5 for Kruskal's Algorithm.

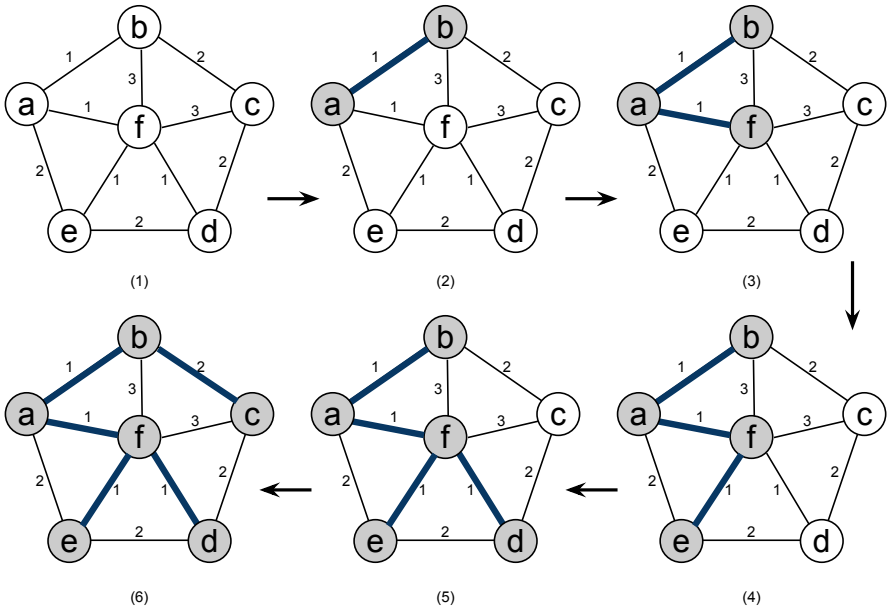

Figure 4: Prim's Algorithm Steps.

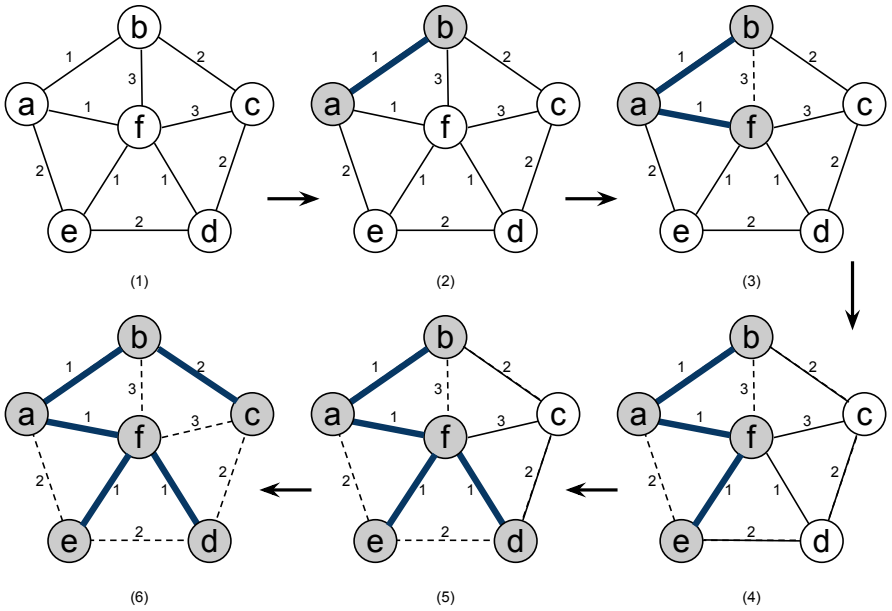

Figure 5: Kruskal's Algorithm Steps.

6. (15 Points). Do heap sort on the input numbers, 10, 21, 7, 15, 4, 25. Start by building a min-heap using **BottomUpHeap()** we learned in the class. Give the intermediate result for each step.
ANSWER:
The heapsort begins by building a heap out of the data set, and then exchange the root(smallest item) with the last item in the array and remove the last item from the heap. After remove the last item, it reconstructs the heap and exchange the smallest item with the last item and do the same thing above. This is repeated until there is only one item left in the heap. Please see result at Figure 6.

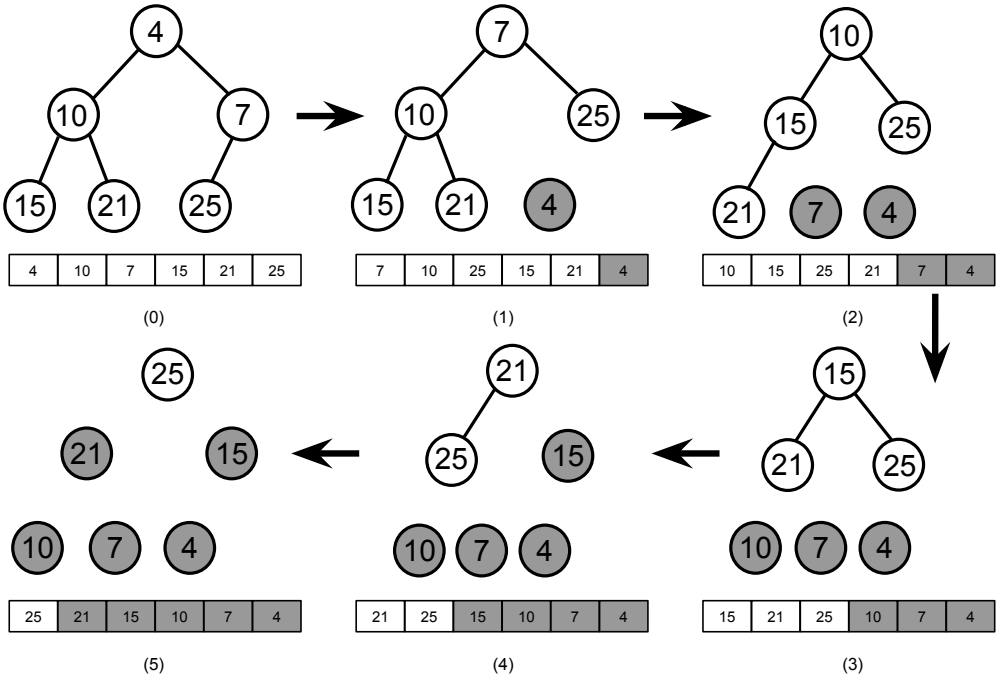

Figure 6: Heap Sort Steps.

7. (15 Points). Assume that we always pick the left most element A[i] in a sequence **A[i ... j]** as the pivot in the QuickSort algorithm **QuickSort(A, i, j)**. Now, assume A is an 8-elements array, containing a permutation of the number $0, 1, \ldots, 7$. Please give one order of elements in A that QuickSort needs $O(n^2)$ to sort it, and an example that QuickSort takes $O(nlog_2(n))$.
ANSWER:
For quick sort, the best case is when each time we perform a partition we divide the list into two nearly equal pieces, so that each recursive call processes a list of half the size. And consequently, we can make only $log_2(n)$ nested calls before we reach a list of size 1. This means the height of call tree is $O(log_2(n))$. And totally the algorithm uses $O(nlog_2(n))$ time.
An example of this case is: 3,1,0,2,5,4,7,6

In the worst case, the two sublists have size 0 and n-1 (already sorted), and the call tree becomes a linear chain of n nested calls (in this case, the height of quick sort tree becomes n-1). And the time for the sort will be $T(n) = O(n^2)$.
An example for this case is: 0,1,2,3,4,5,6,7

8. (20 points). Let A[1..n] be an array of n distinct numbers. If $i < j$, and $A[i] > A[j]$, then the pair (i, j) is called an inversion of A.
a.List all the inversions of the array $< 2, 6, 10, 9, 4 >$. $(2, 5), (3, 4), (3, 5), (4, 5)$.
b.What arrays with elements from the set $\{1, 2, \ldots, n\}$ has the most inversions? How many does it have?
ANSWER:
Array $\{n, n-1, \ldots, 2, 1\}$ has the most inversions. And the number of inversions is:
$n = n - 1 + n - 2 + \ldots + 2 + 1 = \frac{n(n-1)}{2}$.

9. (10 points). Which of the algorithms *Insertion sort, heap-sort, merge-sort*, and *quick sort* are stable? Which are in place?
   ANSWER:
   Insertion sort and merge sort are stable[1].
   Insertion sort and heap sort are in place[2].

[1] Quick sort Can be implemented as a stable sort depending on how the pivot is handled.
[2] Merge sort can be implemented as a stable sort based on stable in-place merging