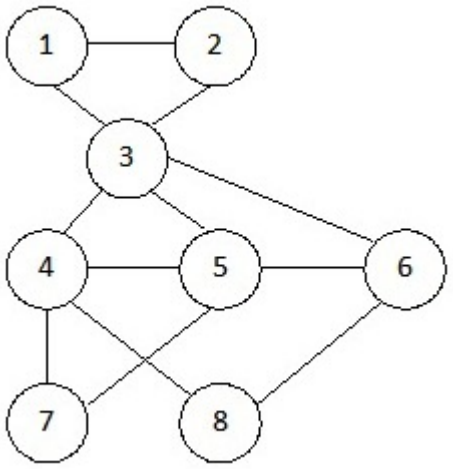
CSI 2110 Fall 2016

**Assignment #5** (50 points, weight 5%)
Due: Wed Dec 7, at 11:59PM on virtual campus.
Late assignments accepted from 1 minute up to 24 hours late with 30% penalty(ie. mark multiplied by 0.7)).
You may type or write by hand legibly and scan it; always submit a single PDF file.

---

**NOTE: you may use the answer template in doc which already has answers tables formatted for you.**

1. (10 points) Let $G$ be an undirected graph whose vertices are integers 1 through 8, and let the adjacent vertices of each vertex be given by the table below:



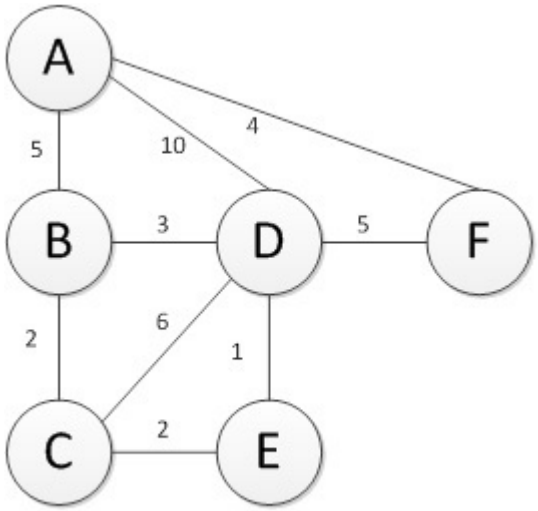

| vertex | adjacent vertices |
|---|---|
| 1 | (2,3) |
| 2 | (1,3) |
| 3 | (1,2,4,5,6) |
| 4 | (3,5,7,8) |
| 5 | (3,4,6,7) |
| 6 | (3,5,8) |
| 7 | (4,5) |
| 8 | (4,6) |

Assume that, in a traversal of $G$, the adjacent lists of each vertex are given as shown in the table above.
NOTE: In general these lists need not be sorted; we are giving them sorted to facilitate your simulation; in any problem of this type you must follow the order given in the corresponding adjacent lists.

(a) (5 points) Give the sequence of vertices of $G$ visited during a Depth-First Search (DFS) traversal starting at vertex 1. Give the list of edges that are marked "discovery" and the list of edges marked "back". Note: edges must be given in the right orientation of the arrows when the labels were given (for example, $\{1, 2\}$ will be either given as $(1, 2)$ or $(2, 1)$ depending if the label was given as an edge when visiting 1 or when visiting 2).

(b) (5 points) Give the sequence of vertices of $G$ visited during a Breadth-First Search (BFS) traversal starting at vertex 1. Give the lists $L_i$ produced by the algorithm. Give the lists of edges that are marked "discovery" and the list of edges marked "cross". Note: edges must be given in the right orientation of the arrows when the labels were given.

2. (8 points) For the graph given in the figure below, use Dijkstra's algorithm to find the shortest path spanning tree of the graph starting from node A.




(a) (7 points) Fill the table below to keep track of the changes of the distance labels after adding each new node into the cloud. The first line of the chart is already filled with the initial distance labels. The first column of the table lists the vertices in order that they join the cloud. The last column of the table will show the edge that was used to include the vertex listed in the first column into the cloud; since we are not showing the updates of the best edge to connect each vertex to the cloud, this is information you need to keep track separately in order to have the correct entry in the last column.

| New vertex | A | B | C | D | E | F | New Edge |
|---|---|---|---|---|---|---|---|
| A | 0 | 5 | $\infty$ | 10 | $\infty$ | 4 | – |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

(b) (1 points) What is the total weight of the shortest-path spanning tree obtained by Dijkstra's algorithm starting at node A?

3. (12 points) Use the same graph as question 2 to calculate the minimum spanning tree using two algorithms.

(a) (6 points) Apply Prim-Jarnik algorithm starting from vertex A.
Give the edges of the minimum spanning tree **in the order** they are inserted into the spanning tree by Prim-Jarnik's algorithm starting at vertex A. The edge will be given by the pair of vertices that connect them (for example, if the edge of weight 5 is selected

out of vertex B, then it should be listed be as "(B,A)"). Give the total weight of the minimum spanning tree.
NOTE: for your own bookkeeping, you could use similar information as you do for Dijkstra algorithm, with the correct notion of distance in the case of Prim-Jarnik's algorithm.

(b) (6 points) Apply Kruskal algorithm. Give the edges of the minimum spanning tree **in the order** they are inserted into the spanning tree by Kruskal's algorithm. Give the total weight of the spanning tree.

4. (10 points) Insert the following keys into a hash table with 11 positions (position 0 through 10), according to the various hash functions and collision resolution techniques specified below.

Insert in this order: 5, 3, 15, 25, 4, 13, 16.

**Example:**

Hash function: $h(k) = k \bmod 11$.
Collision resolution: linear probing.

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| | | 13 | 3 | 15 | 5 | 25 | 4 | 16 | | |

Number of probes to find each key: 5:1, 3:1, 15:1, 25:3, 4:4, 13:1, 16: 4
Average number of probes to find a key: $(1 + 1 + 1 + 3 + 4 + 1 + 4)/7 = 15/7 = 2.14$

For each of the parts below: show the hash table, specify the number of probes needed to search for each of the inserted element after all insertions are performed, and give the average number of probes needed to find a key.

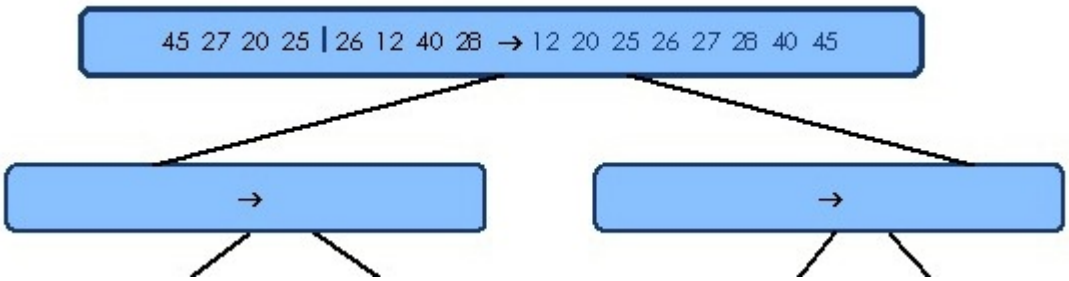
(a) (3 points) Hash function: $h(k) = k \bmod 11$.
Collision resolution: chaining with a separate overflow area.

(b) (3 points) Hash function: $h(k) = k \bmod 11$.
Collision resolution: quadratic probing.

(c) (4 points) Hash function: $h(k) = k \bmod 11$.
Collision resolution:
double hashing with step given by secondary hash function $d(k) = (k \bmod 7) + 1$;
i.e. at step $j = 0, 1, \ldots$ we probe $(h(k) + j * d(k)) \bmod 11$.

5. (10 points) **Sorting Algorithms**

(a) (5 points) Draw the **Mergesort** tree for the following array: a=[45, 27, 20, 25, 26, 12, 40, 28]. Only the nodes for the first partition are show:




Note that if you are typing, you do not need to worry about graphical depiction. Just place the nodes at each level in each line, making it clear which node is parent of which node in the tree. In the example above, you could represent:

```
45,27,20,25|26,12,40,28 -> 12,20,25,26,27,28,40,45
                /                   \
```

Then place kids in each subsequent level into a single line.

(b) (5 points) Suppose that **Quicksort** in-place is used to sort the following array where the pivot is always chosen to be the last number in the subarray being considered. A simplified version of the textbook code fragment 12.6 in page 553 is included here. Show the contents of the array right after "/*** partition step ends ***/" in each recursive call, in order of execution.
For each display show also the contents of parameters a and b (see table that follows).

```
Algorithm quicksortInPlace(int[] S, int a, int b) {
  if (a>=b) return; // subarray is trivially sorted
  /*** partition step starts ****/
  int left=a;
  int right=b-1;
  int pivot=S[b];
  while (left <= right) {
     while ((left <= right) and (S[left]<pivot)) left++;
     while ((left <= right) and (S[right]>pivot)) right--;
     if (left <= right) {
          temp=S{left]; S[left]=S[right];S[right]=temp;
          left++; right --;
     }
  }
  S[b]=S[left]; S[left]=pivot;
  /*** partition step ends ****/
  quicksortInPlace(S,a,left-1);
  quicksortInPlace(S,left+1,b);
}
```

| a | b | S |
|---|---|---|
| - | - | [45, 27, 20, 25, 26, 12, 40, 28] |
| 0 | 7 | |
| . | . | |
| . | . | |
| . | . | |
| . | . | |