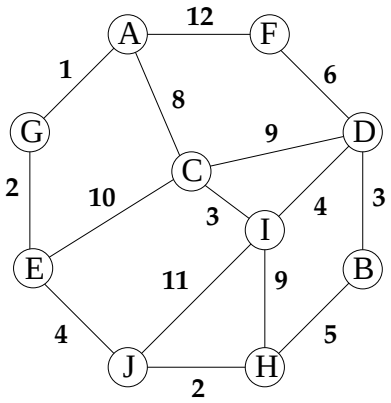
University of Nevada, Las Vegas Las Vegas Computer Science 477/677 Fall 2019

Assignment 6: Due Wednesday October 23, 2019

Name:_____

You are permitted to work in groups, get help from others, read books, and use the internet. But the handwriting on this document must be your own. Print out the document, staple, and fill in the answers. You may attach extra sheets, but only by stapling. Turn in the pages to the graduate assistant at the beginning of class, October 23.

1.



Use Kruskal's algorithm to find the minimum spanning tree of this weighted graph. Use the union/find data structure, and be sure to use path compression. Show the forest after each step. Union/find is explained in section 5.1.4 of the textbook.

2. It is difficult to decide where to put the loop invariant of a **for** loop. However, a **for** loop can always be rewritten as a **while** loop.

   Here is code for the Floyd-Warshall algorithm.

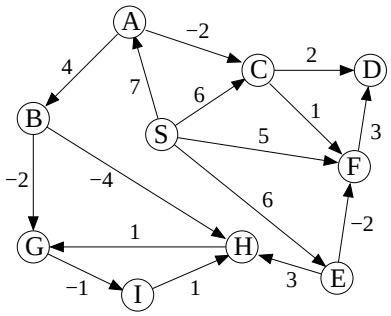
   We assume that G is a directed graph, where $wlu, v)$ is the length (weight) of the edge from u to v. If there is no such edge, we say that $w(u, v) = \infty$. The names of the nodes will be the positive integers $1, \ldots n$. We will construct arrays $cost[u, v]$ for all nodes u and v and $back[u, v]$ for all $u \neq v$.

   Clearly state the loop invariant for the outer loop of this code.

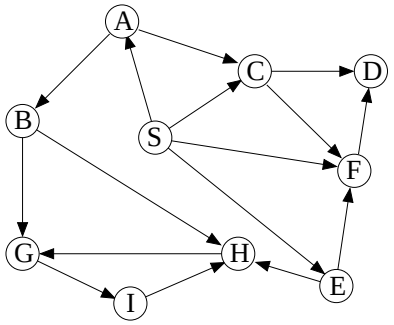
```
bool relax(int&y,int x, int z)
 {
  if( x+z < y)
   {
    y = x+z;
    return true;
   }
  else return false;
 }

void main()
 {
  for(int u = 1; u <= n; u++)
   for(int v = 1; v <= n; v++)
    cost[u,v] = w(u,v);
  for(int u = 1; u <= n; u++)
   for(int v = 1; v <= n; v++)
    if(v != u) back[u,v] = u; // back[u,u] is undefined
  int j = 0;
  // Loop Invariant
  while(j < n)
   {
    // Loop Invariant
    for(int i = 1; i <= n; i++)
     for(int k = 1; k <= n; i++)
      if relax(cost[i,k],cost[i,j],cost[j,k])
       back[i,k] = back[j,k];
    j++;
    // Loop Invariant
   }
  // Loop Invariant
  return 1;
 }
```

3. Let G be the directed graph given on page 120 of your textbook. Work the first step of Johnson's algorithm for the all-pairs shortest path problem on G, *i.e.* adjust the weights so that there are no negative weight edges. Show the adjusted weights in part (b) of the figure below.



(a)



(b)

4. Work Problem 5.18 on page 150 of your textbook, but where we only use the first column of the table. The resulting Huffman code will only encode blank space and the letters e,t,a,o,i,n,s,h.

5. It is well known that the Knapsack problem, explained in Section 6.4 of the textbook, is $\mathcal{NP}$ hard. The Knapsack problem, given in Section 6.4 of your textbook is this: given a set of n items, each of which has a weight and a value, and given a number W (the maximum weight you can carry in your knapsack) find the most valuable set of items whose weight does not exceed W.

   It is well known that the Knapsack problem is $\mathcal{NP}$ hard, which implies that there is no known polynomial time algorithm for that problem. However, there are several variations of the Knapsack problem that can be solved in polynomial time. Here are two of these.

   (a) **Pseudo-Polynomial.** All weights are integers in the range $0..N$. There is a dynamic program given in the textbook which solves that variation of the knapsack problem in $O(nN)$ time. This is not always polynomial in n, since N might not be polynomial in n, but it is polynomial if N is "small," *i.e.* polynomial in n.

   (b) **Powers of 2.** All weights are integers which are powers of 2, There is a polynomial time algorithm for this case, provided the largest weight does not exceed 2^n. To simplify the problem, we can insist that W also be a power of 2, although that restriction simplifies the problem only slightly.

   Suppose $W = 20$ and the weights and values of the items are as given in the table below. Find the optimal set of items by walking through the pseudo-polymial algorithm. Show your work.

weight	value
3	2
5	6
10	7
13	10