University of Houston

COSC 3320: Algorithms and Data Structures
Spring 2016

Solutions for Homework 7

1. Given two strings X and Y, a third string Z is a *common superstring* of X and Y if X and Y are both subsequences of Z. (Example: if $X = \texttt{sos}$ and $Y = \texttt{soft}$, then $Z = \texttt{sosft}$ is a common superstring of X and Y.) Design a dynamic programming algorithm which, given as input two strings X and Y, returns the length of the shortest common superstring (SCS) of X and Y. Specifically, you have to write a recurrence relation $\ell(i,j) = |SCS(X_i, Y_j)|$ that defines the length of a shortest common superstring of X_i and Y_j, and the pseudocode. The algorithm, which has to return $\ell(n,m)$, must run in time $\Theta(n \cdot m)$, where $n = |X|$ and $m = |Y|$. (Hint: use an approach similar to the one used to compute the length of a LCS of two strings.)

   Solution:

   The recurrence relation that defines the length of a shortest common superstring of X_i and Y_j is as follows.

$$\ell(i,j) = \begin{cases} j & \text{if } i = 0, \\ i & \text{if } j = 0, \\ 1 + \ell(i-1, j-1) & \text{if } i,j > 0 \text{ and } x_i = y_j, \\ 1 + \min\{\ell(i, j-1), \ell(i-1, j)\} & \text{if } i,j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

   The pseudocode of the algorithm is as follows.

```
SCS(X,Y)
n = length(X)
m = length(Y)
for i=0 to n do
   L[i,0] = i
for j=1 to m do
   L[0,j] = j
for i=1 to n do
   for j=1 to m do
      if x_i = y_j then
         L[i,j] = 1 + L[i-1,j-1]
      else if L[i-1,j] >= L[i,j-1] then
               L[i,j] = 1 + L[i,j-1]
           else L[i,j] = 1 + L[i-1,j]
return L[n,m]
```

   The complexity of the above algorithm is $\Theta(n \cdot m)$, since a constant amount of basic steps are executed at each of the $n \cdot m$ iterations of the double loop, and the remaining parts of the algorithm have complexity $O(n + m)$.

2. Consider the following simple graph, represented by its adjacency matrix.

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

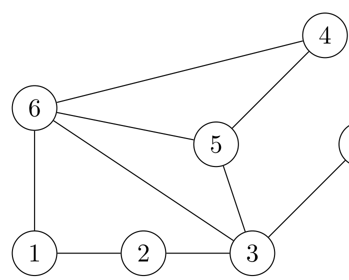
(a) Draw the graph.

(b) Run the DFS algorithm starting from vertex 1, and draw the final DFS tree.

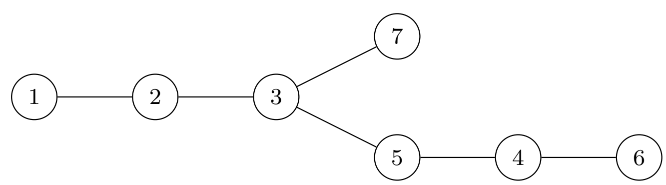
(c) Run the BFS algorithm starting from vertex 1, and draw the final BFS tree.

Solution:

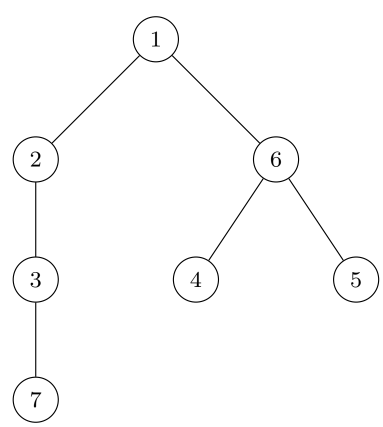
(a)



(b)



(c)



3. Let $G = (V, E)$ be a graph with n vertices and m edges. Design and analyze an algorithm that returns, if it exists, a vertex $i \in V$ such that at least $n/2$ different vertices are reachable, via a path, from i. (Hint: Use the BFS algorithm.)

   Solution:

   Notice that $n/2$ nodes can be reached from a node i if and only if the connected component of i contains at least $n/2+1$ nodes. Hence, the idea is to determine the size of each connected component and, as soon as one of size at least $n/2+1$ is found, return one of its nodes. For each node i of G we can use an additional variable $V[i]$.visited, initially initialized with 0, and modify the BFS algorithm such that it sets the above variable to 1 whenever node i is visited. The pseudocode is as follows.

```
for i=1 to n do
  if (V[i].visited = 0) then
    T <- BFS(G,i)
    if (number of nodes in T >= n/2 + 1) then return i
return null
```

   The algorithm has the same asymptotic complexity of the BFS algorithm, which can be implemented in time $O(n + m)$.

4. Consider the following weighted graph, represented by its adjacency matrix.

$$\begin{bmatrix} 0 & 3 & 0 & 0 & 0 & 4 & 1 \\ 3 & 0 & 10 & 0 & 0 & 0 & 4 \\ 0 & 10 & 0 & 7 & 0 & 0 & 8 \\ 0 & 0 & 7 & 0 & 6 & 0 & 5 \\ 0 & 0 & 0 & 6 & 0 & 5 & 4 \\ 4 & 0 & 0 & 0 & 5 & 0 & 2 \\ 1 & 4 & 8 & 5 & 4 & 2 & 0 \end{bmatrix}$$

   List the edges of the minimum spanning tree in the order they are added by Kruskal's algorithm.

   Solution:

   Assuming vertices are labeled $1, 2, \ldots, 7$, the order is $(1, 7), (6, 7), (1, 2), (5, 7), (4, 7), (3, 4)$.