Search & Sort worksheet -- **KEY**

By Jim Xu

Please work in groups of 2 or 3 to work the following problems. Use additional paper as needed, and staple the sheets together before turning them in. **ONLY TURN IN 1 WORKSHEET/ANSWERS PER TEAM.**

1. [17] Below is an adjacency-matrix representation (denote as $A$) of graph $G$. (empty cell has value 0)

\[
A = \begin{bmatrix}
1 & 2 & 3 & 4 & 5 & 6 \\
1 & & 1 & & & \\
2 & & & & & \\
3 & 1 & 1 & 1 & & \\
4 & 1 & & & 1 & \\
5 & & & & & \\
6 & 1 & 1 & & & \\
\end{bmatrix}
\]

\[
a_{ij} = \begin{cases}
1 & \text{if } (i, j) \in E, \\
0 & \text{otherwise}.
\end{cases}
\]

A. [4] Graph $G$ is a (weighted / unweighted) and (directed / undirected) graph. (circle correct choice)
   (Grading Rubric: 2 points for each)

B. [3] Change $G$ into adjacency-list representation. (Grading Rubric: partial credit was given)

   1. $1 \rightarrow 4$
   2. $3 \rightarrow 2 \rightarrow 3 \rightarrow 5$
   3. $4 \rightarrow 2 \rightarrow 5$
   4. $6 \rightarrow 3 \rightarrow 5$

C. [2] Which of the following represents a correct path? (circle correct choice)
   (Grading Rubric: full credit for only correct choice)

   a. $1 \rightarrow 4 \rightarrow 2 \rightarrow 3$
   b. $3 \rightarrow 5 \rightarrow 6 \rightarrow 3$
   c. $6 \rightarrow 3 \rightarrow 3 \rightarrow 2$
   d. $1 \rightarrow 4 \rightarrow 2 \rightarrow 2$
D. [3] ___2___ vertex(s) has 0 out-degree, how can you find it(them) in adjacency-matrix?
   (Grading Rubric: 2 points for answering number correct, 1 point for explanation)
   Column 2, 5, because they have all 0 in their columns.

E. [3] ___2___ vertex(s) has 0 in-degree, how can you find it(them) in adjacency-matrix?
   (Grading Rubric: 2 points for answering number correct, 1 point for explanation)
   Row 1, 6, because they have all 0 in their rows.

   from $A^T$, compare it with original graph $G$. What is the main difference?
   (Grading Rubric: partial credit was given)
   The $A^T$ reverses all the edges

2. [20] Based on the graph and answer questions below. All the exploring follow alphabetical order.
   Time cost for exploring one node to another node is, say, 1 unit.

A. [3] Directly write down the shortest path from r to x.
   (Grading Rubric: full credit for only correct answer)
   $r -> y -> q -> t -> x$

B. [6] Suppose the start point is q, use the Breath First Search (BFS) algorithm to find a path from q to q. (Use the code below if necessary)
   B1. [5] Write down each step when executing BFS, including the list(queue) you maintained and
   time cost for each node in the list(queue). (Grading Rubric: partial credit was given)
   B2. [1] Is this path a shortest path from q to q? (Grading Rubric: full credit for only correct answer)
   Step1: $q$
   0
   $[q 0], [s 1, t 1, w 1], [t 1, w 1, v 2], [w 1, v 2, x 2, y 2], [v 2, x 2, y 2], [x 2, y 2], [y 2, z 3], [z 3]$
   Note that once q has been removed it cannot be part of a subsequent step
   Yes
C. [5] Suppose the start point is \( r \), use the **Depth First Search (DFS) algorithm** to find a path from \( r \) to \( w \). (Use the code below if necessary)

C1. [4] Write down all the steps as a list and the total time cost. *(Grading Rubric: partial credit was given)*

C2. [1] Is this path a shortest path from \( r \) to \( w \)? *(Grading Rubric: full credit for only correct answer)*

List: 
\[ r \rightarrow u \rightarrow y \rightarrow q \rightarrow s \rightarrow v \rightarrow w, \ 6 \]

No

[2] Comparing BFS and DFS, which one can lead to the shortest path from one node to another? Briefly explain why. *(Grading Rubric: 1 point for correct choice, 1 point for explanation)*

BFS, because BFS searching from smallest step to larger steps, each time we find goal node it will be the smallest step cost.

D. Play with code. (In the code below, s, v, u are vertexes.)

*(Grading Rubric: full credit for only correct answer)*

D1. [2] Change which line (one line) in BFS so that we can control the order of childNode we explore for each parentNode (say, exploring alphabetical order for each node). \[12\]
D2. [2] After which line (one line) in DFS can we insert a printNode() function so that it will print all the nodes we have explored in DFS order?  _____3_____  

D3. [2] After which line in DFS can we insert a printNode() so that it will print all the nodes we have explored in DFS blacking order?  _____8_____

3. [13] Below is a Prerequisite Chart of All Courses in Computer Science Major (From CSU CS website). Based on the Chart and answer the questions below. (View all types of line as required prerequisite)
[4] The Prerequisite Chart of All Courses in Computer Science Major above have valid topological sorts because it has \_directed\_ edge and \_is acyclic\_. (Grading Rubric: 2 points for each)

A. [2] What courses are prerequisites of **CS320 (consider one-level above)**?

(Grading Rubric: full credit for only correct answer)

CS220, MATH161, (MATH229 or MATH 369)
For the followings, (Assume we have studied CS163, CS164, all MATH and STAT courses, so we can view them as existed prerequisites)

B. [2] Write down a **minimum (smallest-numbered)** valid topological sort to **CS454**.
   (Grading Rubric: full credit for only correct answer)
   CS165, CS220, CS320, CS454

C. [3] Which of the following(s) is a valid topological sort? (circle correct choice all that apply)
   (Grading Rubric: full credit for only correct answer) NOTE: b is wrong because although we’ve had 163, 164, and MATH and STAT courses, we haven’t had 165, which is a pre-req for 220
   a. CS165, CS220, CS253, CS270, CS356
   b. CS220, CS270, CS253, CS314, CS453
   c. CS220, CS270, CS320, CS440, CS314
   d. CS165, CS220, CS270, CS370, CS320

E. [2] Play with code. Now we want to implement topological sort algorithm on this Chart in order to find a valid topological sort for all courses. **Fill in the blank below** to make it works as desired.
   (Grading Rubric: full credit for only correct answer)
   if course.inDegree == 0:

```python
class course:
    ""
    Parameters
    ----------
    self.inDegree : int
        The number of edges comes into the course(vertex)
    self.outCourses : List
        A List of courses that dependent on this course
    ""

def topoSort():
    residentCourses = [All courses]
    result = []
    while residentCourses is not empty:
        for course in residentCourses:
            if ________________:
                print("studying ", course)
                for dependentCourse in course.outCourses:
                    dependentCourse.inDegree -= 1
                    result.expand(course)
                residentCourses.remove(course)
    print("a valid topoSort of the Graph is ", result)
```