Before you start:

Homework Files

You can download the starter files for coding as well as this *tex* file (you only need to modify *homework4.tex*) on canvas and do your homework with latex (recommended). Or you can scan your handwriting, convert to pdf file, and upload it to canvas before the due date. If you choose to write down your answers by hand, you can directly download the pdf file on canvas which provides more blank space for solution box.

Submission Form

For homework 4, you need to upload a **pdf file** in the following format:

- VE281_HW4_[Your Student ID]_[Your name].pdf

Please strictly follow the format given above!!! Everyone who does not obey the format will get **2 points** deduction!!!
Notes: No space in your name (use underscore(_) instead), no brackets. One example for name of pdf:
VE281_HW4_518370910000_Run_Peng.pdf

Estimated time used for this homework: **2-3 hours.**

0 Student Info (1 point)

Your name and student id:

Solution: Lan Wang, 519370910084

1 Topology Sort (15 points)

Topology sort is widely used in many different areas. In terms of the implementation of topology sort, actually it is quite similar to what we have learned in the lecture of graph search.

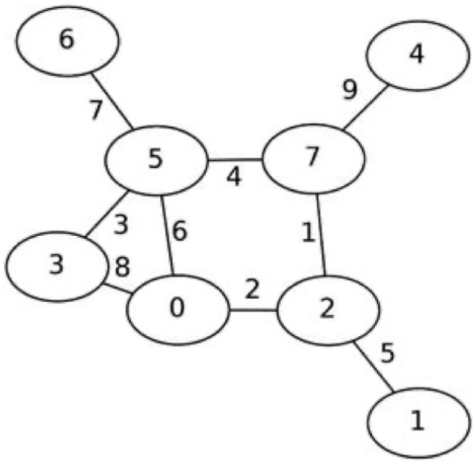
In previous lecture, we have mentioned BFS (Breadth-First Search) and DFS (Depth-First Search). Which one do you think that is similar to the process of topology sort? How can you modify the code of BFS / DFS so that it can be used to do the topology sort? You can show your answer with psuedo code.

Solution:

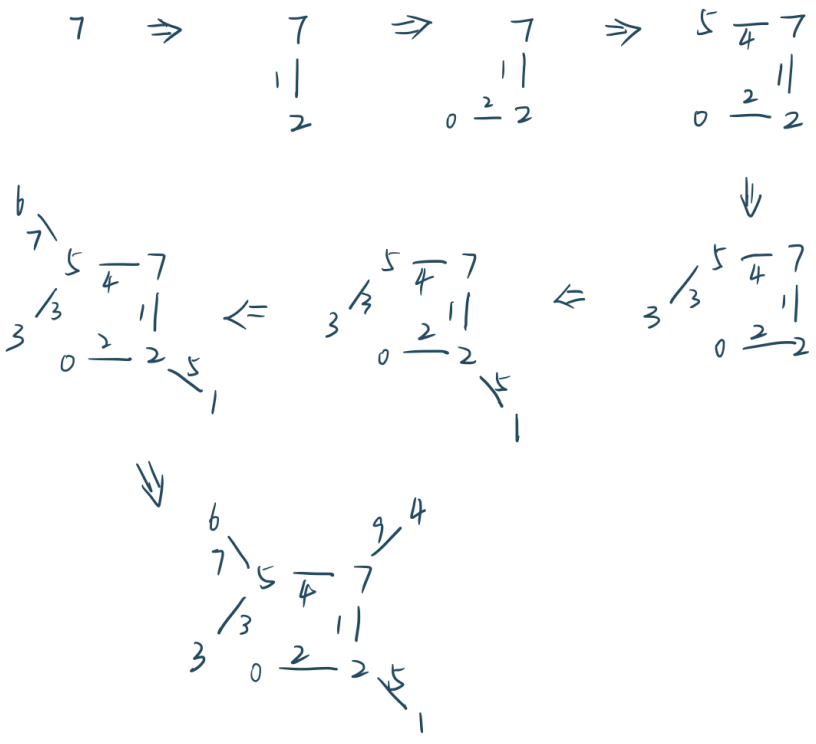
```
topo(s) {
    queue q; // An empty queue
    for(each node u with 0 indegree){
        q.enqueue(u);
    }
    while(!q.isEmpty()) {
        v = q.dequeue();
        for(each node u adjacent to v) {
            decrease indegree of u;
            if(indegree of u equals to 0){
                q.enqueue(u);
            }
        }
    }
}
```

2 MST (24 points)

Starting with vertex 7, plot the MST for the graph above using Prim's algorithm. Please show the intermediate steps (including 8 MST graphs).



Solution:



3 Dynamic Programming (60 points)

All the problems in this part should be written in 15 lines in C++!

3.1 Minimum Nodes in an AVL Tree (20 points)

You are given an AVL tree of height h. Write a function that returns the minimum number of nodes the AVL tree of height h can have, For this problem, **assume that a tree with height 1 only has one node.**

Hint: Since AVL trees must be balanced, the height of each node's left and right children cannot have a difference that exceeds 1.
Example: If h is 3, then your function should return 4, since the smallest possible AVL tree with height 3 has 4 nodes.
Complexity requirement: $O(h)$ time and $O(h)$ auxiliary memory.
You can simply write your answer below. (No need to submit a code file)

Solution:

```
int min_nodes_in_AVL(int h) {
    vector<int>main_nodes_num(h+1,0);
    min_nodes_num.at(0) = 0;
    min_nodes_num.at(1) = 1;
    min_nodes_num.at(2) = 2;
    for(int i = 2; i < h+1; ++i){
        min_nodes_num.at(i) = min_nodes_num.at(i-1) + min_nodes_num.at(i-2) + 1;
    }
    return min_nodes_num.at(h-1);
}
```

3.2 Largest Distance Between Repeated Elements (20 points)

You are given an array with repeated elements. Implement a function that identifies the maximum distance between any two occurrences of a repeated element.

Example: Given the following input:

$$\{1, 2, 3, 2, 2, 1, 3, 3, 2\}$$

Your function should return 7, since the maximum distance between any two repeated elements is the distance between the 2 at index 1 and the 2 at index 8 or 8-1=7.
Complexity requirement: $O(n)$ time and $O(n)$ auxiliary memory, where n is the length of the array. **Hint:** Think carefully about what to store in the auxiliary memory!

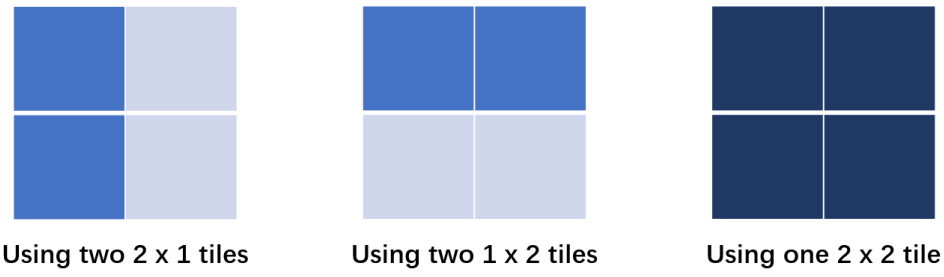
Solution:

```cpp
int max_repeated_distance(const vector<int> &vec) {
    unordered_map<int, int> dis;
    int max_dis = 0;
    for(int i = 0; i < vec.size(); ++i){
        if(dis.find(vec.at(i) == dis.end())){
            dis[vec.at(i)] = i;  // the first index when an element appear
        } else {
            max_dis = max(max_dis,i-dis[vec.at(i)]);
        }
    }
    return max_dis;
}
```

3.3 Tiling (20 points)

Roihn is given a $2 \times n$ board ($n > 0$). He is interested in the number of **distinct** ways to tile the given board using tiles of dimensions (2×1), (1×2), (2×1), (2×2) as shown below:

For example, for a 2×2 (i.e. $n = 2$) board, there are 3 ways:



Complexity requirement: $O(n)$ time and $O(n)$ auxiliary memory

Solution:

```cpp
int number_of_tilings(int n) {
    vector<int> ways_num = (n,0);
    ways_num.at(0) = 1;
    ways_num.at(1) = 3;
    for(int i = 2; i < n; ++i){
        ways_num.at(i) = ways_num.at(i-1)+2*ways_num.at(i-2);
```

```
7       }
8       return ways_num.at(n-1);
9   }
```

Reference

Assignment 4, VE281, FA2020, UMJI-SJTU.