SUBMISSION GUIDELINES

Due date: April 16, 11:59 PM

In Homework 4, we have 5 questions consisting of two types:

1. Short answers. For Q1 and Q2, go to the Gradescope and directly **type your answers** there.
2. Programming. For Q3, Q4, and Q5, you need to submit a .cpp file for each question.
    - Save your function as a separate .cpp file using **the exact function/class name** as its filename (be careful about the upper/lowercase), then upload it to the assignment for the corresponding question on Gradescope.
    - **Function code only**: you only need to keep the required function without any main function, input, or output. You can create more functions as needed, as long as the provided function signature can work as expected.
    - Make sure your code satisfies all the requirements and is robust to corner cases.
    - Do your best to optimize your algorithms (if no specific algorithm is required) rather than use brute force.
    - Code style: your code will be graded by both autograder and TA. Therefore, please make sure that:
        - Your code is **clear enough**. Use blank lines, indent, and whitespace as needed, and delete all the sentences that are no longer needed.
        - Use **understandable variable names**, such as *old_array* and *new_array* rather than *a*, *b*, and *c*.
        - Write necessary **comments** for your key steps. Your code should let others easily understand what it is doing.

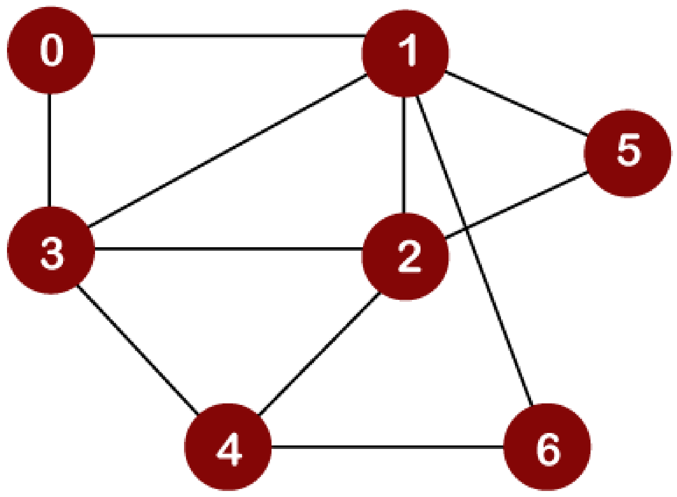
Note:

1. Be honest with yourself. If you could not finish the previous homeworks in 2 days, you probably would not be able to finish this one in 2 days either. The best practice is to start earlier and invest genuine effort, not to cut corners or give up. Grading will be generous as long as you have tried your best. As for plagiarism, it always costs much more than expected as it consists of not only the external outcomes (e.g., failed assignments and courses) but also the burden on your conscience, which you will have to bear.

2. If you have any questions about homework or in-class content, feel free to put your questions in the Discussions on Canvas or show up during the instructor's or TA's office hours. Try everything as best you can instead of trying to find a shortcut or giving up.

Good luck! :D

1. (6 pts) Graph traversals. Consider breadth-first and depth-first search **starting from vertex 0.**



   a. Write the sequence of vertices that would be printed from **breadth-first search**.
   b. Write the sequence of vertices that would be printed from **depth-first search**.

Notes:

- Within each vertex, the edges are traversed in the **increasing order** of adjacent numbers.
- The vertex is **printed** when it is popped from the stack (or removed from the queue).

2. (9 pts) Hashing. The following sequence of keys is added to an initially empty hash table of size 10 with the hash function x mod 10:

   1233, 8549, 8830, 8193, 3657, 9371, 9482, 4499, 4563

   Answer the following questions and explain your answers:

   1. If the table used **separate chaining** to resolve hash collisions, what is the length of the shortest and longest chains in the resulting hash table?
   2. If the table used **linear probing** to resolve hash collisions, what was the maximum number of steps taken to resolve a hash collision?
   3. If the table used **quadratic probing** to resolve hash collisions, what was the maximum number of steps taken to resolve a hash collision?

   Note:

   - Feel free to show your hash table, although it is not required.

3. (11 pts) Path search in a directed graph. Given a directed graph as an adjacency list, write a program to determine if there exists a path between two vertices in the graph.

   Implement the following function:

   ```
   bool pathExists(Graph &g, int vertexFrom, int vertexTo)
   ```

   - Example 1:
     - Input: Graph: {(0,1), (0,2), (1,2), (2,3), (2,0)}, vertexFrom: 0, vertexTo: 3
     - Output: True
   - Example 2:
     - Input: Graph: {(0,1), (0,2), (1,2), (2,3), (2,0)}, vertexFrom: 3, vertexTo: 0
     - Output: False

   Notes:

   - Vertices are indexed with integers starting from zero.
   - You **must** use the following class for graphs:

   ```
   class Graph{
   public:
         int numVertices;
         list<int> *adjLists;
         Graph(int V) {
               adjLists = new list<int>[V];
               numVertices = V;
         }
         ~Graph() { delete[] adjLists; }
         void addEdge(int src, int dest) {
               adjLists[src].push_back(dest);
         }
   };
   ```

   - You will only be queried about vertex indices between 0 and numVertices-1.

4. (14 pts) Dynamic grocery programming. Your parents gave you a credit card to buy some snacks for a party. Feel free to grab some food that is as expensive as possible and can fit into a cart!

   The grocery store provides n items with the volumes v0, v1, ..., vn and prices p0, p1, ..., pn, respectively. You can at most buy the stuff that can fit into a cart with the maximum volume of V. Write a function `fitCart` to figure out the items that you can put in the cart to maximize the price of items.

   - Function signature:
     ```
     vector<bool> fitCart(vector<int>& volumes, vector<int>& prices,
     int maxVolume)
     ```
   - Example:
     - Input:
       - volumes: [1, 1, 2]
       - prices: [3, 4, 2]
       - maxVolume: 3
     - Output: [1, 1, 0]   // Selecting the first two items is the most expensive solution.

   Notes:

   - Volume and price with the same index belong to the same item.
   - All the volumes and prices are greater than 0.
   - You can only select the provided items once. Regard each item as a product on the shelf. In the example above, you can only select the first (price = 3) and the second (price = 4) item once, so the total cost is 3 + 4 = 7. Once selected, it will no longer be available, so the second item cannot be selected twice with the total cost of 4 * 2 = 8.
   - It is possible that several items with the same volume and price are provided at the same time. For example, volumes = [1, 1, 2] and prices = [3, 3, 4] mean that two of the three items are identical, so you can buy at most two of them.
   - You **must** use dynamic programming (**not** top-down recursion) to get the full score.
   - Don't forget to check if an item fits into the cart.

5. (10 pts) Phone book. Implement a class `Contacts` to save, look up, and remove contact names and the corresponding phone numbers.

The functionality expected:

- A phone number should be a **7-digit number** not starting with 0. If a phone number is invalid, do not add it to the phone book.
    - If an input name is invalid or cannot be found, return **-1**.
- A valid name is a **non-empty string**. If a name is invalid, do not add it to the phone book.
    - If an input phone number is invalid or cannot be found, return an **empty string**.
- The adding and removing functions should return **True** if the operation is successful (and the record was added or removed, respectively) – and **False** otherwise.

- Class to implement:

```
class Contacts {
public:
    bool addContact(string name, int number);
    int getNumber(string name);
    string getName(int number);
    bool removeName(string name);
    bool removeNumber(int number);
};
```

- Example:
    - addContact("abc", 1234567); // return True
    - getName(1234567); // return "abc"
    - removeName("cba"); // return False

Notes:

- You **must** use the **map** structure from STL.
- Make sure of the robustness of your code.