The University of Michigan
Electrical Engineering & Computer Science
EECS 281: Data Structures and Algorithms
Fall 2024

Lab 4: Priority Queues, Heaps, and Union Find

Instructions:
Work on this document with your group, then enter the answers on the canvas quiz.

Note:
Be prepared before you meet with your lab group, and read this document so that you know what you must submit for full credit. You can even start it ahead of time and then ask questions during any lab section for help completing it.

You MUST include the following assignment identifier at the top of every file you submit to the autograder as a comment. This includes all source files, header files, and your Makefile (if there is one). If there is not autograder assignment, you may ignore this.

Project Identifier: 15C1680EE94C640EC35E1694295A3625C3254CBA

---

1 Priority Queues & Heaps

1. What does it mean for a tree to be complete?
   A. every node has either 0 or 2 children
   B. every level except the last is necessarily filled, and in the last level, nodes are filled in from left to right
   C. every node in the left subtree must be of lower priority than that of the root, and every node in the right subtree must be of greater priority than that of the root
   D. none of the above

2. Which of the following statements is/are true about a binary heap? Select all that apply.
   A. heapsort has worst-case $\Theta(n \log n)$ time complexity
   B. fix-up has an average-case time complexity of $\Theta(\log n)$
   C. insertion is done by inserting the element at the end, and then calling fix-up
   D. deletion is done by simply removing the root
   E. none of the above

3. Which of the following represents a min-heap? Select all that apply.
   A. [2, 13, 8, 16, 13, 10, 40, 25, 17]
   B. [47, 9, 12, 9, 2, 10, 10, 4, 3, 1]
   C. [3, 5, 6, 7, 12, 15, 14, 9, 10, 11]
   D. [59, 58, 60, 57, 85, 49, 32, 21, 5]
   E. none of the above

4. Consider an empty min-heap priority queue. If you insert the elements 12, 4, 9, 27, 13, 2, and 6 into the heap (in that order, following the algorithm specified in lecture) and remove the most extreme element twice, what are the possible array representations of the heap? Select all that apply.
   A. [6, 9, 12, 13, 27]
   B. [6, 9, 27, 12, 13]
   C. [6, 12, 9, 27, 13]
   D. [27, 13, 12, 9, 6]
   E. none of the above

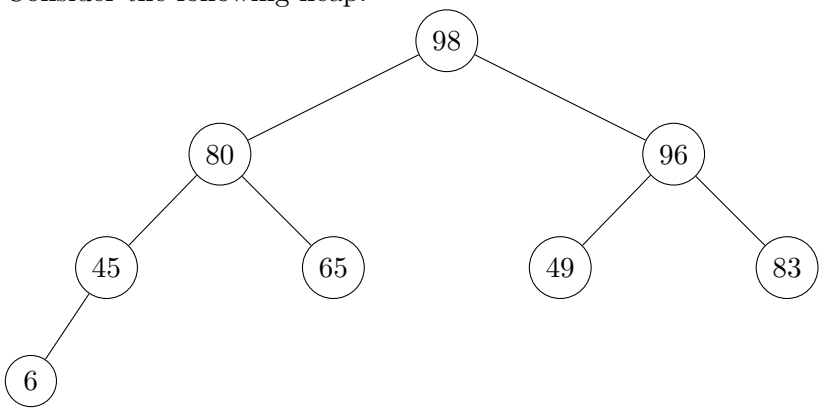
5. Four approaches to heapify are shown below:
   I. Proceeding from the bottom of the heap to the top, while repeatedly calling fixUp()
   II. Proceeding from the bottom of the heap to the top, while repeatedly calling fixDown()
   III. Proceeding from the top of the heap to the bottom, while repeatedly calling fixUp()
   IV. Proceeding from the top of the heap to the bottom, while repeatedly calling fixDown()

   Which of the above approaches will always build a valid heap?
   A. I and II only
   B. I and IV only
   C. II and III only

D. III and IV only

E. I, II, III, and IV

6. Consider the following heap:



You insert the value of 81 into the heap. After the heap invariant is restored, what is the final array representation of this max heap?

A. 98, 80, 96, 45, 81, 49, 83, 6, 65

B. 98, 96, 81, 83, 65, 80, 49, 45, 6

C. 98, 81, 96, 80, 65, 49, 83, 45, 6

D. 98, 81, 96, 80, 65, 49, 83, 6, 45

E. none of the above

7. Consider a min-heap represented by the following array:

[50, 53, 61, 57, 70, 79, 68, 71]

Perform the following operations using the algorithms for binary heaps discussed in lecture. Ensure the heap property is restored at the end of each individual heap operation.

1. Push the value 40 into the min-heap
2. Push the value 54 into this min-heap
3. Update element 71 to have value 49
4. Update element 61 to have value 90

What does the array representation look like after all four operations are completed?

A. [40, 49, 79, 50, 54, 90, 68, 53, 57, 70]

B. [40, 49, 68, 50, 54, 79, 90, 53, 57, 70]

C. [40, 49, 50, 53, 70, 79, 54, 57, 90, 68]

D. [40, 49, 50, 53, 54, 90, 68, 79, 57, 70]

E. None of the above

8. We want to create a k-priority queue, which is a priority queue inspired data structure that allows inserting and removing elements while checking the k-th largest element in O(1) time. What is the optimal time complexity of inserting an element? (**HINT**: think about how you could implement this functionality with two priority queues: a min priority queue and a max priority queue)

A. O(log(n-k))

B. O(2k+log(n-k))

C. O(log(k*k*(n-k)))
D. O(log(k)*log(k)*log(n-k))
E. O(k*k*log(n-k))

9. Consider the following snippet of code:

```
#include <iostream>
#include <queue>
#include <priority_queue>

using namespace std;

class LegoBrick {
    private:
        double width, height;
        string color;
    public:
        LegoBrick(double width_in, double height_in)
            : width(width_in), height(height_in) {}

        void printLegoBrick() const {
            cout <<    (    << width << "," << height << ") ";
        }

        const double getArea() {
            return width * height;
        }
};

struct LegoBrickCompare {
    bool operator()(LegoBrick& a, LegoBrick& b) const {
        return a.getArea() < b.getArea();
    }
};

int main()
{
    vector<LegoBrick> bag_o_legos = { LegoBrick{1,2}, LegoBrick{2,4},
    LegoBrick{1,1}, LegoBrick{4,2},
    LegoBrick{3,3}, LegoBrick{1,5},
    LegoBrick{2,2},  LegoBrick{2,5}};

    priority_queue<LegoBrick, vector<LegoBrick>, LegoBrickCompare>
        my_fav_legos(bag_o_legos.begin(), bag_o_legos.end());

    for(int i = 0; i < 3; ++i){
        my_fav_legos.top().printLegoBrick();
        my_fav_legos.pop();
    }
}
```

You are searching through an old bag of legos and want to pick your three favorite pieces. What is the output to the code above?

A. (1,1) (1,2) (1,5)
B. (3,3) (4,2) (2,4)
C. (3,3) (2,4) (4,2)
D. (2,5) (3,3) (1,5)
E. (2,5) (3,3) (4,2)

2 Union Find

10. Consider the union-find container given below and suppose you used path compression when implementing union-find.

Item	1	2	3	4	5	6	7	8	9	10
Representative	1	8	1	2	5	4	6	10	3	5

I. After calling find on item 7, 4's representative would be 8.

II. After calling find on item 6, 8's representative would be 5.

III. Calling find on 3 would change one or more representatives in the table.

IV. There are 2 disjoint sets represented here.

V. After calling find on item 2, 6's representative would be 5.

Which of the following is true given these 10 elements and their representatives?

A. I only

B. II, IV

C. IV only

D. I, III, IV

E. II, IV, V

3 Handwritten Problem

This problem is to be submitted independently. We recommend trying it on your own, checking your answer with your group and discussing solutions, and then submitting at the end of lab. These will be graded on completion, not by correctness. However, we want to see that you were thinking about the problem. The starter files can be found on Canvas.

11. Given n ropes, find the minimum cost of connecting ropes where the cost of connecting two ropes is the sum of their lengths.

    Example: If you had 4 ropes of lengths 10, 5, 8, 11, the min cost is 68

    Explanation: 1. Join ropes of length 5 and 8 to get rope of length 13 (Net cost = 13) 2. Join ropes of length 10 and 11 to get rope of length 21 (Net cost = 13 + 21) 3. Join ropes of length 13 and 21 to get rope of length 34 (Net cost = 13 + 21 + 34 = 68)

    Another example: If you had 10, 5, 8, 14, the min cost is 73.

    Explanation: Join ropes 5 and 8 to get a rope of length 13 (Net cost = 13) Join ropes 13 and 10 to get a rope of length 23 (Net cost = 13 + 23) Join ropes 23 and 14 to get a rope of length 37 (Net cost = 13 + 23 + 37 = 73)

    ```
    int join_ropes(const vector<int>& rope_lengths);
    ```


4 Coding Assignment

12. Recall the union-find data structure that was covered in this lab. Union-find has many practical applications, so knowing how to implement this algorithm will certainly come in handy. In this lab, you will be using the union-find algorithm to solve the following problem.

    A graph is a collection of vertices and edges (we will cover graphs in more detail after the midterm). We call a graph connected if there exists a path between every pair of vertices. Similarly, a connected component is a subgraph that is connected, and where no vertex in the subgraph has an edge to a vertex outside the subgraph. For example, in this illustration, there are three connected components:

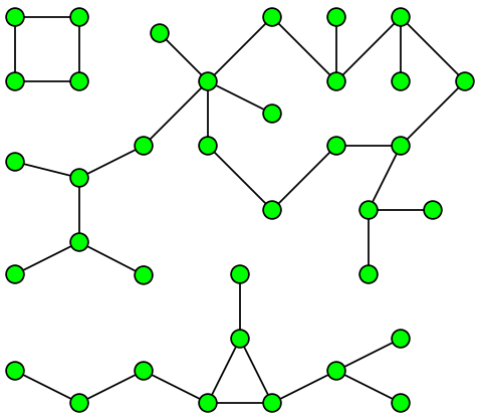
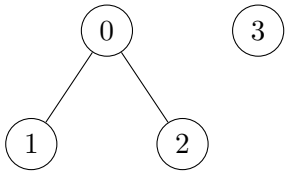


Image Source: https://en.wikipedia.org/wiki/Connected_component_(graph_theory)

You are given the scaffolding of a graph abstract data type in `connected_components.cpp`, and a `main()` that reads in and constructs a graph given the edges listed in a file. You may retrieve the files from Canvas or from the EECS 281 GitHub (https://github.com/eecs281staff/l04-connected-components). Your task is to write a program that calculates and prints out the number of connected components in the graph represented by the file. For your convenience in understanding and testing, the input file is formatted as follows, listing each of the edges in the graph:

```
<number of vertices> <number of edges>
<start vertex> <end vertex>
<start vertex> <end vertex>
... // etc ... As many as <number of edges> lines
```

For example, the following input:
4 2
0 1
0 2
describes a graph that looks like the one shown below. Your program would print out 2 in this case.



Submitting to the Autograder

Make sure you write your program in the `connected_components.cpp` file. You will be able to make three submissions per day. To submit to the autograder, use the Makefile to generate a .tar.gz file. Simply run the following command in the directory of the `connected_components.cpp` file and Makefile:

```
make fullsubmit
```

If you are working with a partner, make sure that both of you submit to the autograder. Only students who submit code to the autograder will receive points. Both of you can submit the same code for this assignment.

Make sure you include the assignment identifier on all code files you submit to the autograder.

Testing

You will be given three test case files on Canvas. The solutions to these three test cases will not be released in this document. However, if you get any of these cases wrong when you submit to the autograder, the autograder will tell you the solution you got, as well as the correct solution you should have gotten. Thus, it is in your best interest to make your first submission early.

To get an idea of what answers you should be getting for these three test cases, you should expect outputs that fall in the range [1, 10]. If you get anything outside of this range, then you are doing something incorrectly.

However, you may not assume that the answers to all 18 test cases on the autograder will fall into the range [1, 10]. In fact, several of these cases will produce answers that are quite large. In addition, you cannot assume that there is a limit to the number of vertices or edges you will be given. The test cases that are run on the autograder may have vertex or edge counts in the tens of millions.

Since the cases we provide publicly are still quite large, it is important that you also write smaller test files of your own, especially if you aren't passing the ones that are given. This will simplify the debugging process and will make your life easier. An additional test case is provided below.

To actually run a test, simply use the Makefile provided to make an executable. This executable will be named connected. Running the `test_case1.in` test, for example, can be done using the command:

```
./connected < test_case1.in
```

Additional Test Case

Here is an additional test case for debugging:

```
20 15
7 3
7 1
9 2
2 1
1 3
2 4
7 8
16 12
17 11
17 14
```


14 16
12 10
10 11
11 16
18 13

should produce a graph with 8 connected components:

0
1, 2, 3, 4, 7, 8, 9
5
6
10, 11, 12, 14, 16, 17
13, 18
15
19

