**CS233 Data Structures and Systems Programming**

Homework 3 - Spring 2017 - 40 Points

DUE @ TAPS on Lesson 37, M-day: Wed, 26 April, T-day: Thurs, 27 April.

***Documentation Policy:***

* You must document all help received from any source other than your instructor.
* The documentation statement must explicitly describe WHAT assistance was provided. WHERE on the homework the assistance was provided, and WHO provided the assistance.
* If no help was received on this assignment, the documentation statement must state “NONE.”
* If you checked answers with anyone, you must document with whom on which problems. You must document whether or not you made any changes, and if you did make changes you must document the problems you changed and the reasons why.
* Vague documentation statements must be corrected before the assignment will be graded, and will result in a 5% deduction on the assignment.

***Help policy for Homework #3:***

**AUTHORIZED RESOURCES: Any, except another cadet’s work or programs.**

**NOTE:**

* Never copy another person’s work and submit it as your own.
* Do not jointly complete this assignment.
* You must document all help received from sources other than your instructor.
* **DFCS will recommend a course grade of F for any cadet who egregiously violates this Help Policy or contributes to a violation by others.**

**Instructions:** Type your answers into this document and **Git** 🡪 **add** this document to the Homeworks folder of your *source code repository*. Make sure you *commit* and *push* your repository before the due date. (The file name should be **black** (not red or green) after you have submitted the document.)

**Documentation Statement:** None

***Hash Table* Data Structure (10 pts)**

1. (10 pts) Assume you are implementing a hash table. Your hash table is defined using an array of 100 elements and it stores strings. For each of the following functions, state whether it would be a reasonable hash function for your hash table. If it could not be used as a hash function, **say why**.
   1. (3 pts)  
      int hashFunction1(HashTable \* table, void \*data) {  
       return rand() % table->arraySize;  
      }

This could not be used as a hash function since it returns a different value every time. It would never be able to find the correct index from a key.

* 1. (3 pts)  
     int hashFunction2(HashTable \* table, void \*data) {  
      char \* str = (char \*) data;  
      return (str[0] << 24) + (str[1] << 16) + (str[2] << 8) + str[3];  
     }

This hash function assumes each string is at least 4 characters long, and also would lead to some very large indices given the large shift amount. It would work, but would not necessarily be efficient.

* 1. (4 pts)  
     int hashFunction3(HashTable \* table, void \*data) {  
      char \* str = (char \*) data;  
      int index = str[0] << 8 + str[4];  
      return index % table->arraySize;  
     }

This hash function assumes each string is at least 5 characters long, but would not lead to creating excessively large indices. However, the array size is a dynamic variable, and thus the hash function would not return the same value for each key. Therefore, this is not a valid hash function.

***Heap* Data Structure (10 pts)**

1. (5 pts) You have the following array of integers, where element[0] is not used.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Value | - | 34 | 18 | 5 | 1 | 12 | 56 | 37 | 4 | 15 | 6 | 8 | 62 | - | - | - |

Assume that you convert the array above to a max-heap using the following algorithm:

void convertToMaxHeap(Heap \* heap) {

int n = heap->numberElements;

// Note that the leaf nodes don't need to be shifted down,

// so we start at the parent of the last node.

for (int root = n / 2; root >= 1; root--) {

shiftDown(heap, n, root);

}

}

For the resulting max-heap, assume that given element k, its parent is at index (k/2) and its children are at indexes (2\*k) and (2\*k + 1). Show in the diagram below what the contents of the array will be after it becomes a max-heap.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Value | - | 62 | 18 | 56 | 15 | 12 | 34 | 37 | 4 | 1 | 6 | 8 | 5 | - | - | - |

1. (5 pts) You have the following max-heap that contains integer elements, where element[0] is not used. Assume that given element k, its parent is at index (k/2) and its children are at indexes (2\*k) and (2\*k + 1).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Value | - | 72 | 10 | 22 | 8 | 9 | 7 | 21 | 3 | 1 | 4 | 5 | 6 | - | - | - |

Assume that you insert a new integer, 37, into the max-heap using the following algorithm:

void maxHeapInsert(Heap \*heap, int \*element) {

// Add the element to the end of the array

heap->array[heap->numberElements+1] = \*element;

shiftUp(heap, heap->numberElements, heap->numberElements+1);

}

Show in the diagram below what the contents of the array will be after the new element, 37, is inserted.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Index | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| Value | - | 72 | 10 | 37 | 8 | 9 | 22 | 21 | 3 | 1 | 4 | 5 | 6 | 7 | - | - |

***Graph* Data Structure (20 pts)**

1. (8 pts – 2 pts each) All of the following graph diagrams have the same nodes, but the edges are configured differently. Please answer the questions below and make sure you include a rationale for your answers. No rationale 🡪 no points.

|  |  |  |
| --- | --- | --- |
| (1) | (2) | (3) |

1. Which graph diagram represents a ***list*** data structure? **And why?**

#2 represents a list, because there is only one way to navigate the nodes

1. Which graph diagram represents a ***tree*** data structure? **And why?**

#1 represents a tree, because each node has a parent (excluding the root) and one or more children (excluding leaf nodes), and there are no other edges (i.e. cross, forward, back)

1. Which graph diagram represents a ***set*** data structure? **And why?**

#3 represents a set, because it is just a grouping of nodes, none of them are connected to each other

1. If a graph can represent sets, lists, and trees, why don’t we use graphs as a single "universal" data structure and forget about sets, lists and graphs?

These more specific data structures are better oriented to organize certain data and manipulate it correspondingly.

1. (6 pts – 3 pts each)   
   Given the graph,

3

7

2

1

8

4

3

* 1. Show its computer representation if it was stored using an adjacency matrix (2D array).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **0** | **1** | **2** | **3** | **4** | **5** |
| **0** | 0 | 2 | 4 | 3 | 8 | 0 |
| **1** | 2 | 0 | 0 | 1 | 0 | 7 |
| **2** | 4 | 0 | 0 | 0 | 3 | 0 |
| **3** | 3 | 1 | 0 | 0 | 0 | 0 |
| **4** | 8 | 0 | 3 | 0 | 0 | 0 |
| **5** | 0 | 7 | 0 | 0 | 0 | 0 |

* 1. Show its computer representation if it was stored using an adjacency list (a linked list of the nodes a node is connected to).

4

8

NULL

Data

Weight

Next

3

2

0

1

2

2

3

3

2

4

1

5

7

NULL

2

0

2

2

3

1

2

2

3

4

3

NULL

2

1

1

NULL

2

0

3

2

0

4

2

4

1

7

NULL

2

0

8

2

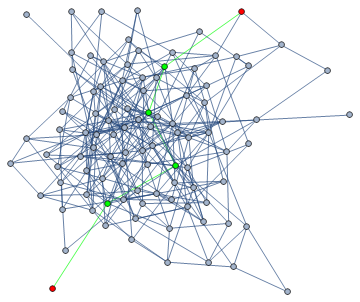
4

3

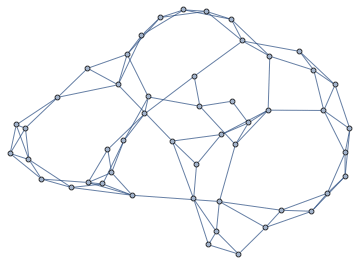
NULL

2

5

1. (2 pts) For the graph to the right, which representation (*adjacency matrix* or *adjacency list*) should be used to store it in the computer **and why?**

An adjacency matrix should be used because the graph is relatively complete. A matrix would make the most effective use of memory.

1. (2 pts) For the graph to the right, which representation (*adjacency matrix* or *adjacency list*) should be used to store it in the computer **and why?**

An adjacency list should be used because the graph is sparse, a list would make a more efficient use of memory.

1. (2 pts) Explain how memory must be allocated in a C program to create a 2D array at run-time using dynamic memory allocation, (i.e., using malloc) so that the elements of the 2D array can be easily accessed using the C language "double bracket" notation. For example, use arrayName[2][3] to access the element of the second row and the third column.

A pointer pointer must be used to create a 2D array in C. A malloc function would be called to create a row. Then, another malloc function must be called to make the array, allocating memory based on the number of columns, multiplied by the size of the rows.