Homework 5

100 Points

Heaps / Graphs

Choose one of the following projects:

5.1 Selection Sort and Heap Sort
5.2 Priority Queue
5.3 Huffman Encoding Algorithm (using a heap) // Choose this assignment for extra credit
Requirements: - Array representation of the heap
- Implement **`reheapUp`** and **`reheapDown`** as recursive functions
5.4 Graphs – Solve a maze problem. // Choose this assignment for extra credit

→ 5.1 Selection Sort and Heap Sort

Create an array of random numbers within the range -99 to 99 of a given size. Sort it using the selection sort algorithm. Sort the same array using the heap sort algorithm. If the size of the array is less than or equal to 100, print the arrays before and after sorting (for both selection and heap sort). Change both algorithms to keep track of the number of data comparisons and data moves. Save these results. Repeat the process with other arrays. Then write short report to compare the results. Suggested arrays sizes for testing:
10 10 100 100 1000 1000 10000 10000 100000 100000.

→ 5.2 Priority Queue

An airline company uses the formula shown below to determine the priority of the passengers on the waiting list for overbooked flights.

Priority number = A / 1000 + B – C

Where

A is the customer's total mileage in the past year
B is the number of years in her or his frequent flier program
C is the sequence number representing the customer's arrival position when s/he booked the flight (the first customer's sequence number is 1, second in the file is 2, and so on).

Build the heap based on the serial number: serial number = priority * 100 + (100 – C)
Given a file with overbooked customers, **`overbooked.txt`**, write a program that reads the file and determines each customer's priority number and prints a list of waiting customers (name and their priority and serial numbers) in priority sequence, including the number of customers.
A line in the input file contains the number of years in the frequent flier program, total mileage in the past year, and the name of the customer (see below)

```
5 53000 Robert Hill
3 89000 Amanda Trapp
3 93000 Jonathan Nguyen
5 53000 Tom Martin
1 17000 Mary Lou Gilley
3 89000 Bob Che
7 72000 Warren Rexroad
```

```
2 65000 Vincent Gonzales
3 34000 Paula Hung
6 21000 Lou Masson
4 42000 Steve Che
3 89000 Linda Lee
3 63000 Dave Lightfoot
5 53000 Sue Andrews
2 33000 Joanne Brown
7 99000 Paul Ng
5 53000 Steven Chen
2 65000 Vladimir Johnson
7 72000 Peter Edwards
```

→ 5.3 Huffman Encoding Algorithm (using a heap)

// Choose this assignment for extra credit

First calculate the frequency of each character in the text given below. Create a min-heap of binary trees (frequency is the key). Each tree in the heap has only one node that contains the character and its frequency. Then start building the Huffman tree:

loop (as long as the heap has two or more nodes)

- remove two binary trees from the heap (smallest an second smallest frequency)
- merge the two trees: the root of the merged tree stores the sum of the frequencies
- insert the merged tree into the heap

end loop

When done, traverse the tree to generate the Huffman codes (as a strings "0000") and save the Huffman codes into a hashed table. Display the Huffman codes. Use the hashed table to encode the text below. Use the Huffman tree to decode the text.

> In computer science, a data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to specific tasks. For example, B-trees are particularly well-suited for implementation of databases, while compiler implementations usually use hash tables to look up identifiers. Data structures are used in almost every program or software system. Data structures provide a means to manage huge amounts of data efficiently, such as large databases and Internet indexing services. Usually, efficient data structures are a key to designing efficient algorithms. Some formal design methods and programming languages emphasize data structures, rather than algorithms, as the key organizing factor in software design. Data structures are important because the way the programmer chooses to represent data significantly affects the clarity, conciseness, speed of execution, and storage requirements of the program.

→ 5.4 Graphs – Solve a maze problem.

// Choose this assignment for extra credit

The graph is another structure that can be used to solve the maze problem. Every start point, dead end, goal, and decision point can be represented by a node. The edges between the nodes represent possible paths through the maze. Write a program that simulates a mouse's movement through the maze using a graph and a depth-first traversal. When the program is complete, print the path through the maze. Use the following two graphs to test your program. Create another input file of your choice.

maze1.txt

```
7       // number of nodes in the graph: 0 to 6
6       // starting point
3       // goal
0 2     // edges: node 0 and node 2 are connected by an edge
0 5
1 2
1 4
1 6
2 3
4 5
```

maze2.txt

```
15
0
14
0 1
1 2
1 4
2 3
2 6
4 5
4 6
5 6
5 7
7 8
7 9
9 10
9 11
11 12
11 13
11 14
```

Note: here is another way to store a maze (graph) in a file. Choose the one that you prefer.

```
7       // number of nodes in the graph: 0 to 6
6       // starting point
3       // goal
0 2 5   // edges: node 0 and node 2 are connected by an edge, 0 and 5
1 2 4 6  // edges: 1 – 2, 1 – 4, 1 – 6
2 3
4 5
```