Dept. of Computer Science
Trent University

COIS 2020H: Assignment 2
Heap Matrices, Open Hash Tables, and Huffman Codes
(100 marks)

Due Sunday, November 09, 2025 at 23:59
A late penalty of 10% per day is assessed until Friday, November 14, 2025 at 23:59
No assignment can be accepted after Friday, November 14, 2025 at 23:59

Part A: Heap Matrices (30 marks)

Background

The binary heap seen in class is implemented as linear array and can be used to represent an instance of the class Priority Queue. An alternate way to represent a Priority Queue is an m x n matrix where:

1) the items of each row are sorted in decreasing priority from left to right, and
2) the items of each column are sorted in decreasing priority from top to bottom.

The matrix therefore can hold at most m x n items. If there are fewer items, then the remaining locations are set to undefined (e.g. +∞ or null). We will call this special type of matrix a Heap matrix.

Example of a 4 x 5 Heap matrix with 14 integers:

17	33	67	87	99
25	42	70	91	+∞
45	53	78	+∞	+∞
85	93	+∞	+∞	+∞

To consolidate your background on the Heap matrix, review the following questions. Assume that P is an m x n Heap matrix with zero-based indices (like C#). Argue that:

1) P has no items if P[0,0] is undefined.
2) P has m x n entries if P[m-1,n-1] is defined.
3) the number of items in row k is greater than or equal to the number of items in row k+1.
4) the number of items in column k is greater than or equal to the number of items in column k+1.
5) the item with the highest priority is found at P[0,0].

Requirements

Using the Heap matrix as its representation, re-implement the class Priority Queue with the following methods. For each method, state its time complexity using the big-Oh notation in terms of m and n.

```
class PriorityQueue<T>
{
        private T[,] H;                 // Heap matrix
        private int numRows;            // Number of rows
        private int numCols;            // Number of columns

        // 1 mark
        // Initialize an m x n Heap matrix
        public PriorityQueue ( int m, int n ) { … }

        // 6 marks
        // Insert an entry into the priority queue
        public void Insert (T item ) { … }

        // 6 marks
        // Remove the item with the highest priority (6 marks)
        public void Remove ( ) { … }

        // 1 mark
        // Return the item with the highest priority (1 mark)
        public T Front ( ) { … }

        // 4 marks
        // Return true if the item is found in the priority queue; false otherwise (4 marks)
        public bool Found (T item) { … }

        // 2 marks
        // Return the number of items in the priority queue (2 marks)
        public int Size ( ) { … }

        // 2 marks
        // Output the Heap matrix (2 marks)
        public void Print ( ) { … }
}
```

Mark Breakdown for Part A

PriorityQueue Methods	22
Time complexities	2
Testing	6

Part B: Open Hash Table Revisited (25 marks)

Background

In class, I made the comment that it may be advantageous to maintain the keys in each bucket in sorted order. For example, when a new <key,value> pair is inserted into the hash table, it is necessary to check that the key is not already present. If the linked list associated with a bucket is sorted, the search can be curtailed once either the key is found or the key in the linked list is greater than the key to be inserted. Hence, it is not always necessary to traverse the entire linked list.

Requirements

Re-implement the open hash table seen in class with the following modifications.

1. Each bucket has a header node.
2. The <key,value> pairs in each bucket are sorted by TKey.
3. One additional method called Output returns all <key,value> pairs in sorted order based on their keys.

```
public void Output( ) { … }
```

To test your implementation, create a small class called Point whose integer data members, x and y, represent the coordinates of a point in two-dimensional space. The Point class will serve as TKey. Therefore, ensure that Point inherits from IComparable and overrides the methods GetHashCode and Equals. For the CompareTo method, assume that points are ordered first on the x-coordinate and secondarily on the y-coordinate.

Mark Breakdown for Part B

class Point	
CompareTo	2
GetHashCode	1
Equals	1
class HashTable header	1
MakeEmpty	1
Insert	4
Remove	2
Retrieve	2
Output	6
Testing	5

Part C: Huffman Codes (40 marks)

Background

In 1952, David Huffman published a paper called *"A Method for the Construction of Minimum-Redundancy Codes"*. It was a seminal paper. Given a text of characters where each character occurs with a frequency greater than 0, Huffman found a way to encode each character as a unique sequence of 0's and 1's such that the overall length (number of bits) of the encoded text was minimized.

Huffman recognized that characters with a greater frequency should have shorter codes than those with a lower frequency. Therefore, to disambiguate two codes, no shorter code of 0's and 1's can serve as the prefix (beginning) of a longer code. For example, if letter 'a' has a code of 010, no other character can have a code that begins with 010.

To find these codes, a Huffman tree is constructed. The Huffman tree is a binary tree where the left edge of each node is associated with the number 0 and the right edge is associated with the number 1. All characters are stored as leaf nodes and the code associated with each character is the sequence of 0's and 1's along the path from the root to the character.

Requirements

As a start, read up on Huffman Codes in your text. Then, implement the Huffman class which tackles each of the following tasks.

Task 1: Determine the Character Frequencies
Determine the frequency of each character in the given text and store it in an array using the character itself to map to an index. Assume all printable ASCII characters, codes 32 to 126, may be used as part of the text (see AnalyzeText).

Task 2: Build the Huffman Tree
Using a priority queue of binary trees, a Huffman tree is constructed for the given text. Initially, the priority queue is populated with as many as 95 individual nodes (binary trees of size 1) where each node stores a character and its frequency. The priority queue is ordered by frequency where a binary tree with a lower total frequency has a higher priority than one with a higher total frequency (see Build).

Task 3: Create the Codes
The final Huffman tree is then traversed in a prefix manner such that the code for each character is stored as a string of 0s and 1s and placed in an instance of the C# Dictionary class using its associated character as the key (see CreateCodes) and the string as the value.

Task 4: Encode
Using the dictionary of codes, a given text is converted into a string of 0s and 1s (see Encode).

Task 5: Decode
Using the Huffman tree, a given string of 0s and 1s to converted back into the original text (see Decode).

To help implement your program, consider the following skeleton of code.

```
class Node : IComparable
{
        public char     Character       { get; set; }
        public int      Frequency       { get; set; }
        public Node     Left            { get; set; }
        public Node     Right           { get; set; }

        public Node (char character, int frequency, Node left, Node right)  { … }

        // 2 marks
        public int CompareTo ( Object obj )  { … }
}

class Huffman
{
        private Node HT;                        // Huffman tree to create codes and decode text
        private Dictionary<char,string> D;      // Dictionary to store the codes for each character

        // Constructor
        // Invokes AnalyzeText, Build and CreateCodes
        public Huffman ( string S )  { … }

        // 4 marks
        // Return the frequency of each character in the given text
        private int[ ] AnalyzeText ( string  S )  { … }

        // 10 marks
        // Build a Huffman tree based on the character frequencies greater than 0
        private void Build ( int[ ] F ) { PriorityQueue<Node>  PQ; … }

        // 8 marks
        // Create the code of 0s and 1s for each character by traversing the Huffman tree
        // Store the codes in Dictionary D using the char as the key
        private void CreateCodes ( )  { … }

        // 4 marks
        // Encode the given text and return a string of 0s and 1s
        public string Encode ( string S  )  { … }

        // 6 marks
        // Decode the given string of 0s and 1s and return the original text
        public string Decode ( string S )  { … }
}
```

Mark Breakdown for Part C

CompareTo	2
Huffman methods	32
Testing	6

For Parts A, B and C

Testing

Testing is an indispensable part of software development and builds confidence in the robustness of your program. For this assignment:

1) Create a document in .pdf that itemizes the test cases for each method of your implementation. This document can be prepared even **before** your program is implemented.

2) Using your test document, ensure that your code works correctly for all methods and for erroneous input. You may use screens shots for your testing.

Suggestion: Test your implementation as it is written, not as one large program at the end. When a method is implemented, test it right away. Ensure that it works before moving on. This gives you confidence that your code is correct to that point.

Source Code Documentation and References (5 marks)

Your source code should be readable and well-commented. Keep in mind that choosing informative variable names is a form of source code documentation.

Hints

1. Form your team quickly.
2. Begin the assignment ASAP. Even setting up the classes as presented in this assignment is a good way to get started.
3. Draw diagrams on a separate sheet of paper to help you design each of the algorithms.
4. Compile and test your methods incrementally (one at a time).
5. Develop test cases even before you begin to program.
6. Document as you program. It helps your team members to understand your code.

Submission

Assignments are to be completed in teams of 2 or 3 members. Solo assignments or assignments with more than 3 team members will not be accepted. Please refer to our Discussion Board on Blackboard to connect with fellow classmates.

Zip up your project (all files including the executable), the test documentation and results, and submit your assignment online at Blackboard. Only one team member needs to submit the assignment but do make sure that **all** names are on the submission.