Algorithm Analysis

Name: Maiqi Hou, Course: CSI 3344

1.

Description

Creating a priority queue

Data Structure

A array and queue

Algorithm

```
For i = 0 to size (Set a size arbitrarily according to the ASCII code)
    If array[ASCII Code] not equal zero
        Push into the priority queue
```

Analysis

Input N	Array’s size

Basic Operation	void priorityQ(int array[]) {
	for (int i = 0; i < 128; i++) { // n = 128
	if (array[i] != 0) { // n
	Node* n = new Node(i, array[i]); // n
	pq.push(n); // n
	}
	}
	}

Summation or Recurrence Relation	T(n) = O(n)

Worst Case Analysis

T(n) = O(n)

Best Case Analysis

T(n) = O(n)

2.

Description

Concerte a Huffman Tree

Data Structure

Priority queue

Algorithm

Priority queue q

While q’s size remians 1

    Declare Node left = q’s top, then pop.

    Delcare Node right = q’s top, then pop.

    Declare new Node = (left node’ weight plus right node’s weight)

    New node’s right and left equal to Node left and Node right

    Push new node into q

Analysis

Input N	Priority queue’s size

Basic Operation	while (q.size() != 1) {
	Node* left = q.top();
	q.pop();
	Node* right = q.top();
	q.pop();
	Node* node = new Node(128, left->weight + right->weight);
	node->left = left;
	node->right = right;
	q.push(node);
	}

Summation or Recurrence Relation	T(n) = O(log n)

Worst Case Analysis

T(n) = O(log n)

Best Case Analysis

T(n) = O(log n)

3.

Description

Record the code of the character

Data Structure

Map and array

Algorithm

```
If root’s left  exists
        array [index]= 0
        Recurrsive root’s left, index +1
If root’s right exists
        Array = 1
        Recurrsive root’s right,index +1
If root’s right and left not exists
        Map [root’s character] = array
```

Analysis

Input N	Number of node = n

Basic Operation	`void huffcode(Node* t, map<int, string>& m, int index, char a[])`
	`{`
	`        if (t->left) {`
	`            a[index] = '0';`
	`            huffcode(t->left, m, index+1,a);`
	`        }`
	`        if (t->right) {`
	`            a[index] = '1';`
	`            huffcode(t->right, m, index+1,a);`

	}
	if (!t->left && !t->right) {
	    string chara;
	    for (int i = 0; i < index; i++) {
	        chara += a[i];
	    }
	    m[t->ch] = chara;
	}
	
	}

Summation or Recurrence Relation	T(n) = O(n^2)

Worst Case Analysis

T(n) = O(n^2)

Best Case Analysis

T(n) = O(n)

4.

Description

Pass the map to main, storing each character's code

Data Structure

map

Algorithm

None

Analysis

Input N	None

Basic Operation	void Encode(map<int, string>& table) {
	        table = m;
	    }

Summation or Recurrence Relation	T(n) = 1;

Worst Case Analysis

T(n) = 1

Best Case Analysis

T(n) = 1

5.

Description

Read compressed file, read the character's counts

Data Structure

array

Algorithm

For i = 0 to n = 128(Seting size according to ASCII code)

Analysis

Input N	N = Seting size according to ASCII code

Basic Operation	void Store(int array[], string s, ofstream& a)
	for (int i = 0; i < 128; i++) {
	if (array[i] != 0) {
	a << i << ” ” << array[i] << endl;

Summation or Recurrence Relation	T(n) = O(n)

Worst Case Analysis

T(n) = O(n)

Best Case Analysis

T(n) = O(n)

6.

Description

Recreate huffman tree

Data Structure

priority queue

Algorithm

While file not terminate

File read int type character and number of character

Push into a priority queue

Analysis

Input N	Number of file’s lines

Basic Operation	void ReCreateTree(string filen) {
	ifstream f;
	f.open(filen);
	int index = 0, n = 0, ti = 0;
	while (!f.eof()) {
	f >> index >> n;
	Node* node = new Node(index, n);
	pq.push(node);
	}
	ti = index;
	}
	}

Summation or Recurrence Relation	T(n) = O(log n)

Worst Case Analysis

T(n) = O(log n)

Best Case Analysis

T(n) = O(log n)

7.

Description

Codes decode each character

Data Structure

None

Algorithm

```
For i = 0 to string s’ size
        If s[i] = ‘0’
                Node point to node’s left
        Else
                Node point to node’s right
        If node’s right and left not exists
                Return character
```

Analysis

<table>
<tr><td>Input N</td><td>String’s size = n</td></tr>
<tr><td>Basic Operation</td><td>

```
void Decode(string s, string& c) {
    for (int i = 0; i < s.size(); i++) {
                if (s[i] == ’0’) {
                    temp = temp->left;
                }
                else if (s[i] == ’1’){
                    temp = temp->right;
                }
                if (!temp->left && !temp->right) {
                    c += char(temp->ch);
                    temp = pq.top();
                }
            }
```

</td></tr>
</table>

	}
Summation or Recurrence Relation	T(n) = n

Worst Case Analysis

T(n) = O(n)

Best Case Analysis

T(n) = O(n)