

Assignment 6 Technical Report

2171039 Chaewon Lee

Code 1

This code randomly generates the element to be sorted and sorts the elements by heap sort.

To do this, first, size of heap and output is allocated as the size of the input. Then, since heap sort includes the process that the elements of the array are inserted into the heap, main function starts to measure the time.

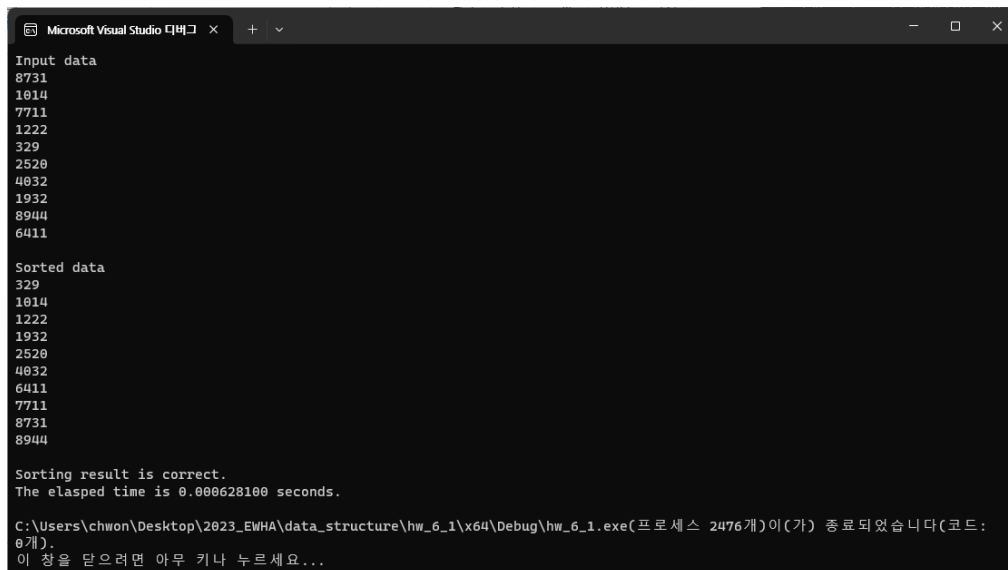
Then, elements are randomly generated and inserted into the heap. Then `heap_sort` function performs the heap sort. Since the half of the elements already meet the heap property, we only have to check $n/2 \sim 1$ element whether they need to be moved or not. To do this, I used the code from `delete_max_heap` function because `delete_max_heap` function also checks the child node whether it is smaller than its parent or not, and when it is, then moves the parent node. If the parent node is i , then the smallest child node's number would be $i*2$. The code picks which is smaller between two child nodes of the parent. And parent node and smaller child node would be swapped. This would be continued until child node gets bigger than the size of the heap.

After construction the max heap, by using `delete_max_heap` function, `heap_sort` function takes all of the elements out of the max heap and gets the sorted result. `Delete_max_heap` function takes out the first element of the heap, which is the biggest element in the heap, and consider the element of the smallest number as the first element. Then, it keeps moving while there is no smaller element down the heap. Since the first element of max heap must be the biggest element in the heap, the result of the deletion would be sorted in increasing order.

Then, the time measurement stops. `Check_sort_results` function checks whether the elements are sorted or not by checking whether the two neighboring elements are in increasing order. Finally, main function prints the time spent in operating heap sort.

If the input size is smaller than 20, main function prints both elements inside the heap and sorted data. And since this takes extra time, if you want to check the pure time spent in heap sort, you should remove this printing part.

- Result



```
Microsoft Visual Studio 디버그 콘솔
Input data
8731
1014
7711
1222
329
2520
4032
1932
8944
6411

Sorted data
329
1014
1222
1932
2520
4032
6411
7711
8731
8944

Sorting result is correct.
The elapsed time is 0.000628100 seconds.

C:\Users\chwon\Desktop\2023_EWHA\data_structure\hw_6_1\x64\Debug\hw_6_1.exe(프로세스 2476개)이(가) 종료되었습니다(코드:
0개).
이 창을 닫으려면 아무 키나 누르세요...
```

You can see the input data is randomly generated, and it is sorted in appropriate order. And also, you can check the time spent in sorting.

Code 2

This code generates the Huffman code of each character by Huffman coding and encodes and decodes the given string.

First, data and freq array is set as the data and frequency of huff1, which stores data about Huffman code. And the size of huff1, which means the count of characters in the given string, is assigned as m_char_size, which can be modified in line 48.

2D array m_LUT saves the each codeword of the character, and m_bit_size saves the length of the each codeword. Main function initializes these arrays to use them after.

Then, Huffman_traversal function traverses the tree and generates the Huffman codeword for each character. This function gets node, parent, and level as the argument. If the node is left child of the parent, 0 must be added in the last of the code. If the node is the right child, then 1 must be added. Then, the bit_size of node will be one longer than its parent. If node has data, which means the node points the character of the given data array, m_LUT and m_bit_size should be set to save the Huffman code of each character. Since characters given in this code is a to f, in alphabetical order, I just subtracted 'a' to get the index of the given character, but if it is not the case, you should linearly find which index is exactly the same with the character of data you want to assign.

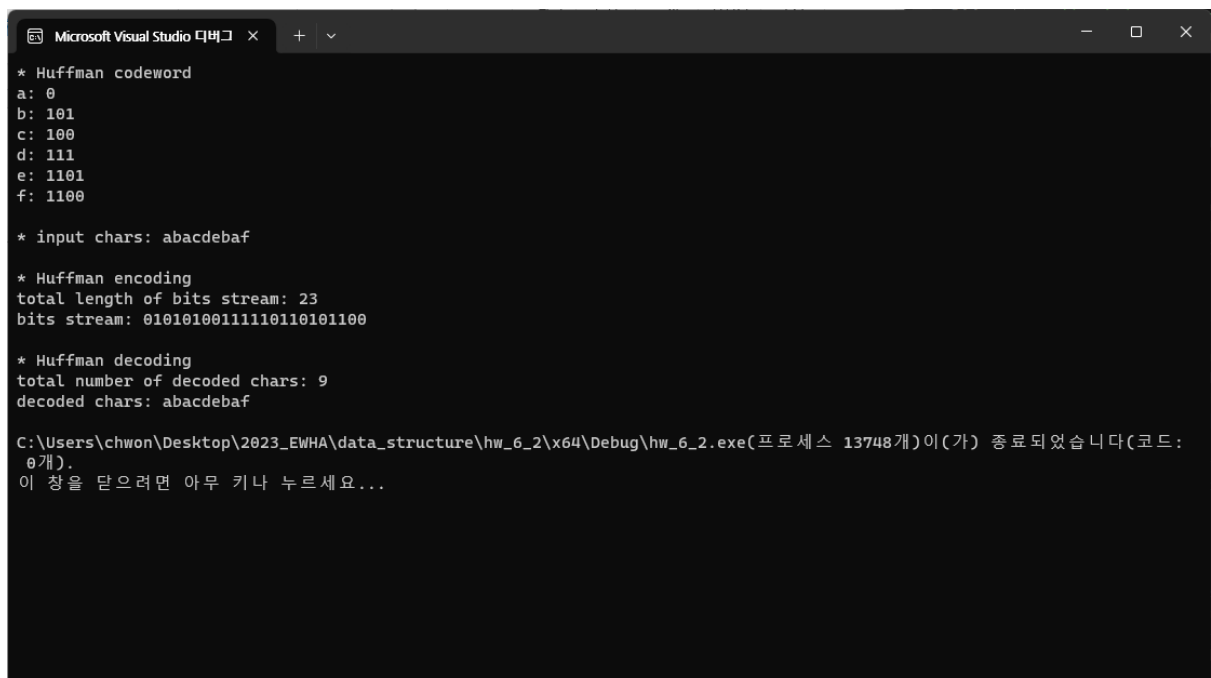
If there is no child node, then the code stops traversing. If it is not, code keep traverses its left and right node with increasing its level.

After generating the code, print_codeword function prints the each codeword for the

character. Then, Huffman_encoding function encodes the given string, 'str' with the codeword generated above. In Huffman_encoding function, there is a for loop that checks str[i] and if it is NULL, it breaks. Then it adds the Huffman codeword of corresponding str[i]. To find the corresponding index of m_LUT, I also subtract 'a' here, but if data is not in an alphabetic order, you should find it linearly. After the process, the function prints the bit stream that the original string is converted as.

In Huffman_decoding function, this decodes the generated bit stream into original string. First, it sets the root node as the current node, and keep tracking until index variable is smaller than the total length of the bit stream. If the number of the bit stream is 0, we should check the left side. And if it is 1, we should check the right side. But there is no child node in the direction that we want to check, then we should stop traversing, and decode that codeword as the character. We can consider the character of that code as the data inside of the current node. Then, we set the current node as the root node of the tree and keep traversing until reaches the end of the bit stream.

- Result



```
Microsoft Visual Studio 디버그 콘솔
* Huffman codeword
a: 0
b: 101
c: 100
d: 111
e: 1101
f: 1100

* input chars: abacdebaf

* Huffman encoding
total length of bits stream: 23
bits stream: 01010100111110110101100

* Huffman decoding
total number of decoded chars: 9
decoded chars: abacdebaf

C:\Users\chwon\Desktop\2023_EWHA\data_structure\hw_6_2\x64\Debug\hw_6_2.exe(프로세스 13748개)이(가) 종료되었습니다(코드: 0x00000000).
이 창을 닫으려면 아무 키나 누르세요...
```

You can see that each character is converted into codeword according to their frequency well, and both encoding and decoding process is also performed well.