Assignment 3
2015313754 TaehyungGil (길태형)

November 15, 2020

My code consists of three parts.

1. Get the whole input sequences from ‘hw3_input.txt’.

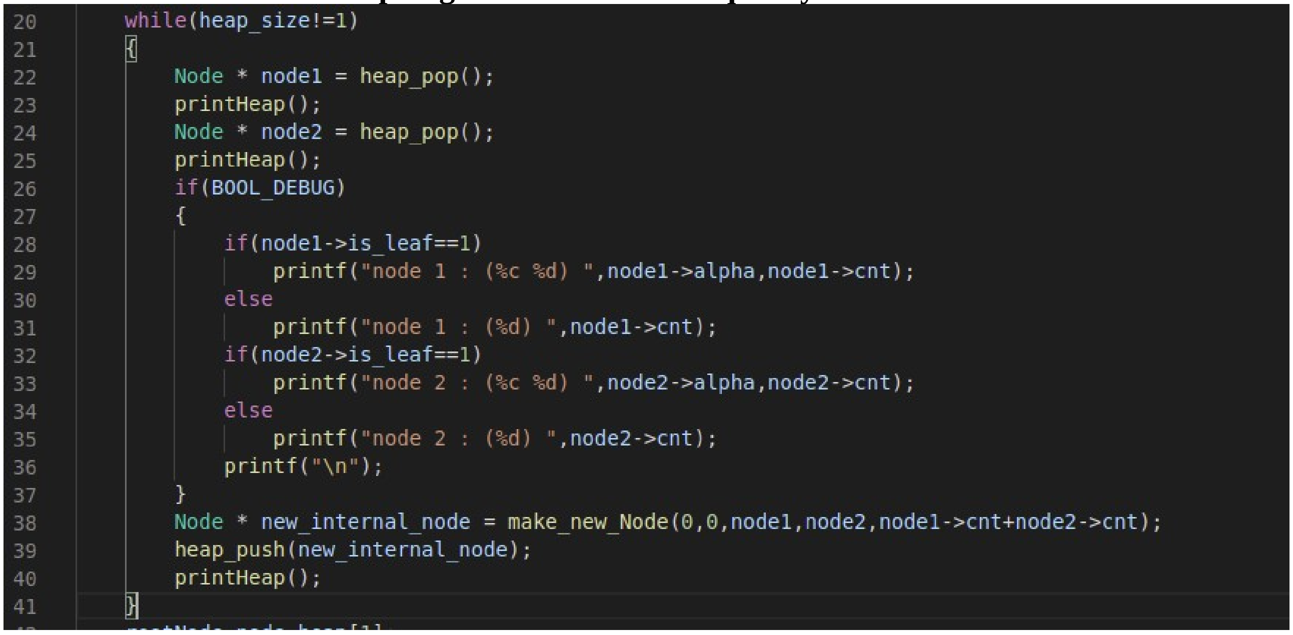
2. Make hw3_output1.txt

3. Make hw3_output2.txt

Below are detailed explanations and performance analysis of each part.

1. Get the whole input sequences from ‘hw3_input.txt’.

It reads input file and count the frequency of each alphabet character.
Using the frequency information, make huffman tree.
I used minimum heap to get the minimum frequency nodes.

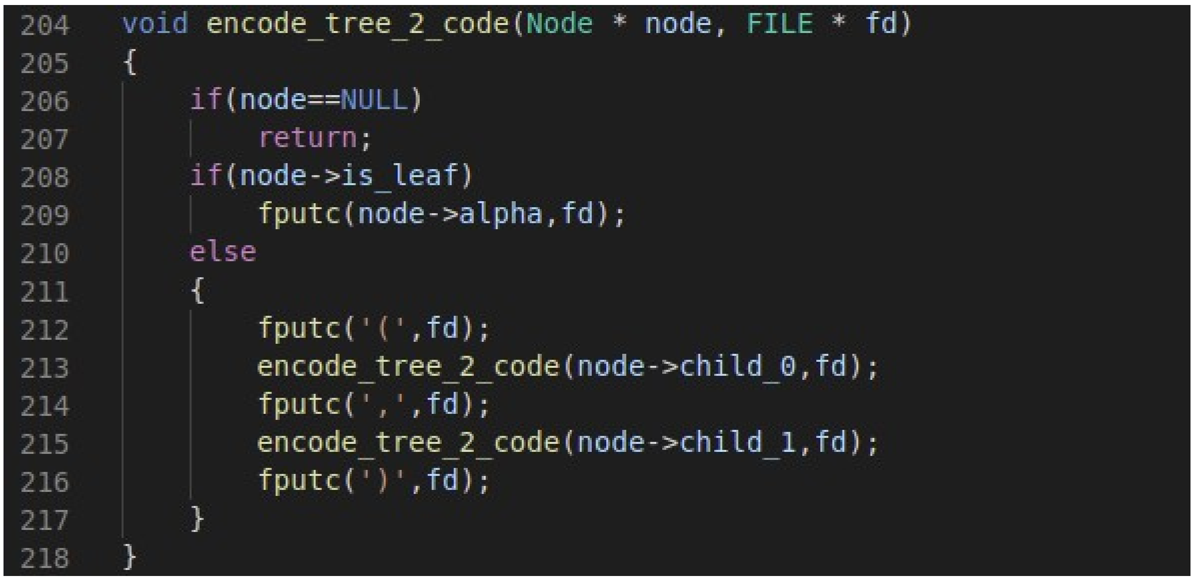


```
    while(heap_size!=1)
    {
        Node * node1 = heap_pop();
        printHeap();
        Node * node2 = heap_pop();
        printHeap();
        if(BOOL_DEBUG)
        {
            if(node1->is_leaf==1)
                printf("node 1 : (%c %d) ",node1->alpha,node1->cnt);
            else
                printf("node 1 : (%d) ",node1->cnt);
            if(node2->is_leaf==1)
                printf("node 2 : (%c %d) ",node2->alpha,node2->cnt);
            else
                printf("node 2 : (%d) ",node2->cnt);
            printf("\n");
        }
        Node * new_internal_node = make_new_Node(0,0,node1,node2,node1->cnt+node2->cnt);
        heap_push(new_internal_node);
        printHeap();
    }
```

Using this, I made huffman tree.

2. Make hw3_output1.txt

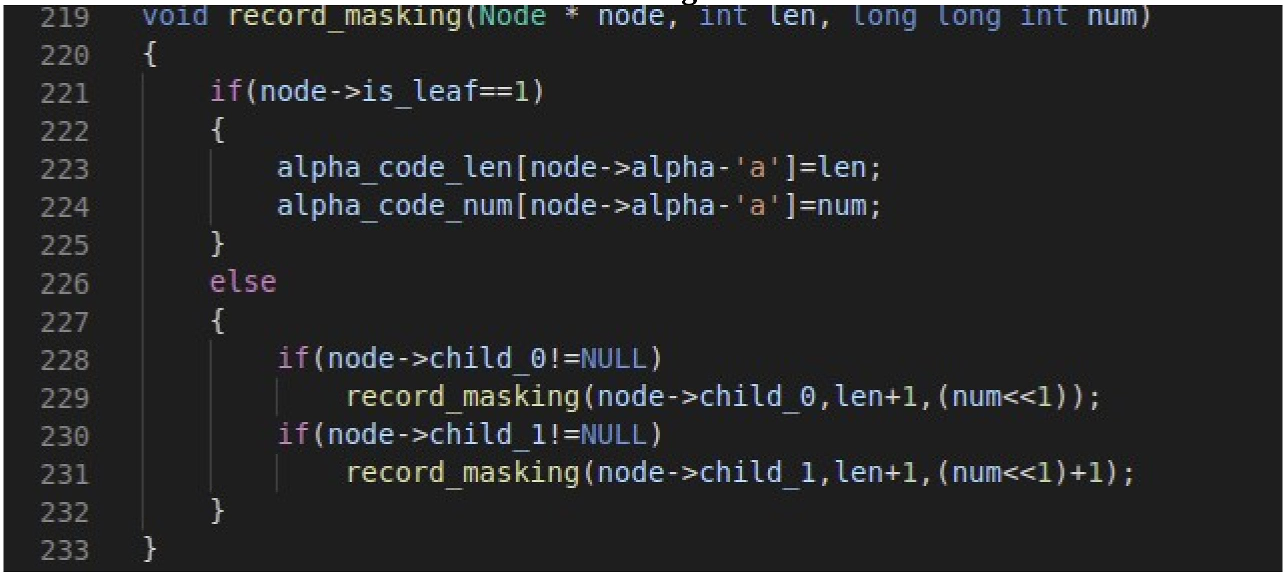
After making huffman tree, I changed the tree to string.



```c
void encode_tree_2_code(Node * node, FILE * fd)
{
    if(node==NULL)
        return;
    if(node->is_leaf)
        fputc(node->alpha,fd);
    else
    {
        fputc('(',fd);
        encode_tree_2_code(node->child_0,fd);
        fputc(',',fd);
        encode_tree_2_code(node->child_1,fd);
        fputc(')',fd);
    }
}
```

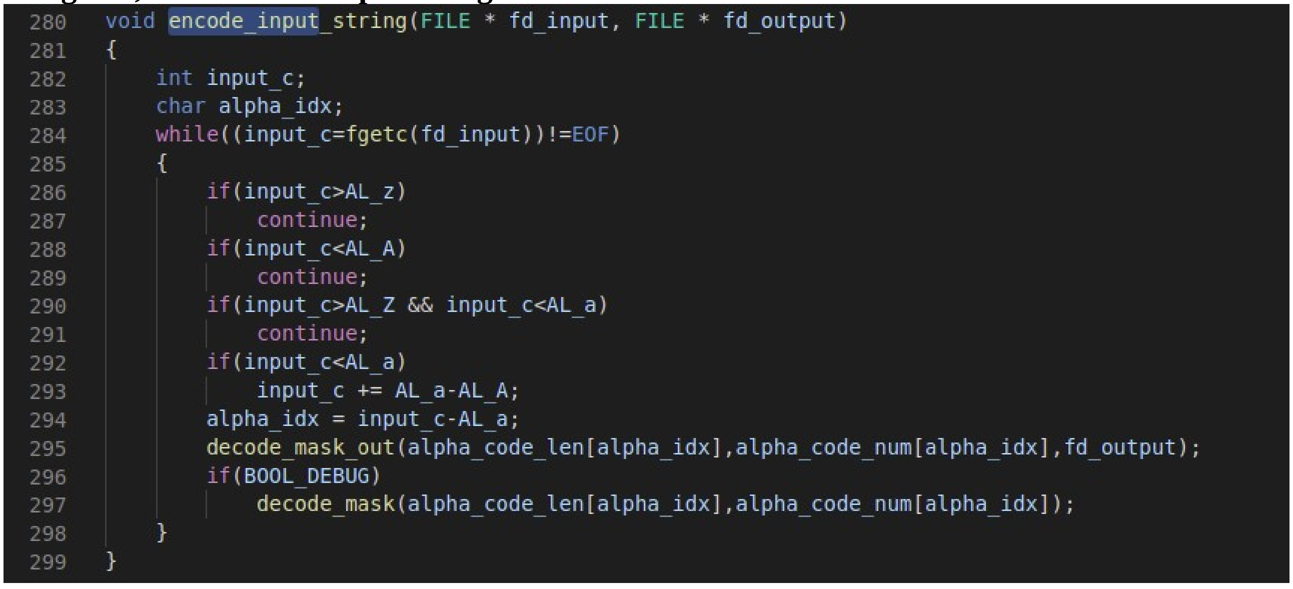
Pretraveling the tree, if the node is an internal one, it puts '(', calls left child, puts ',', calls right child, and puts ')'.

And I recorded each leaf node's huffman code length and value.



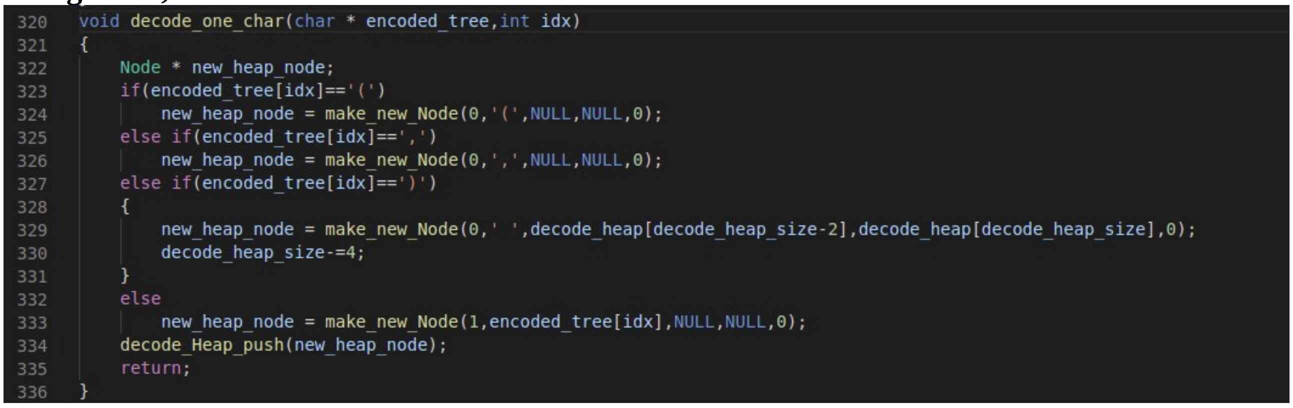
```c
void record_masking(Node * node, int len, long long int num)
{
    if(node->is_leaf==1)
    {
        alpha_code_len[node->alpha-'a']=len;
        alpha_code_num[node->alpha-'a']=num;
    }
    else
    {
        if(node->child_0!=NULL)
            record_masking(node->child_0,len+1,(num<<1));
        if(node->child_1!=NULL)
            record_masking(node->child_1,len+1,(num<<1)+1);
    }
}
```

using this, I encode the input string.



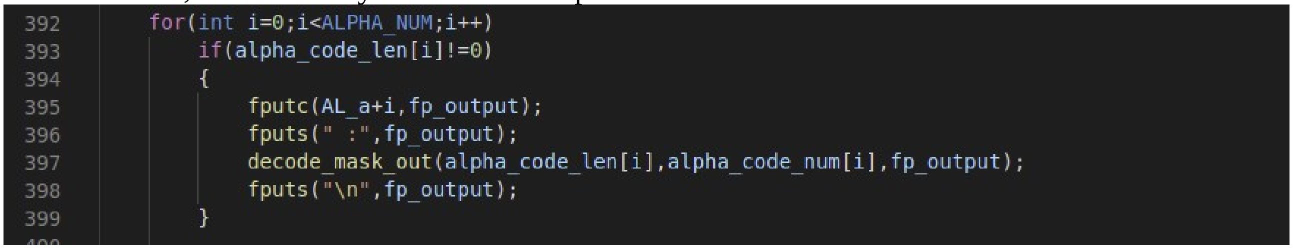
```c
void encode_input_string(FILE * fd_input, FILE * fd_output)
{
    int input_c;
    char alpha_idx;
    while((input_c=fgetc(fd_input))!=EOF)
    {
        if(input_c>AL_z)
            continue;
        if(input_c<AL_A)
            continue;
        if(input_c>AL_Z && input_c<AL_a)
            continue;
        if(input_c<AL_a)
            input_c += AL_a-AL_A;
        alpha_idx = input_c-AL_a;
        decode_mask_out(alpha_code_len[alpha_idx],alpha_code_num[alpha_idx],fd_output);
        if(BOOL_DEBUG)
            decode_mask(alpha_code_len[alpha_idx],alpha_code_num[alpha_idx]);
    }
}
```

3. Make hw3_output2.txt
Using stack, I reconstruct the huffman tree.



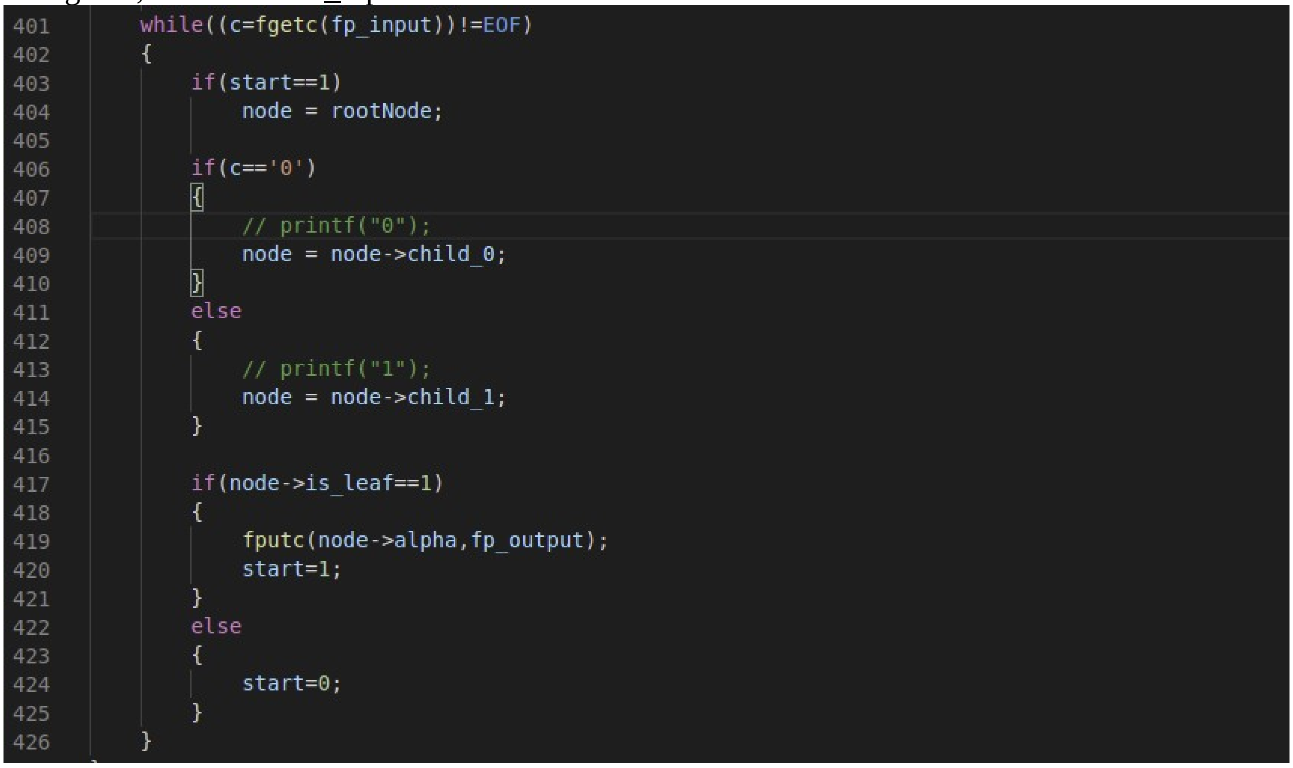
```
void decode_one_char(char * encoded_tree,int idx)
{
    Node * new_heap_node;
    if(encoded_tree[idx]=='(')
        new_heap_node = make_new_Node(0,'(',NULL,NULL,0);
    else if(encoded_tree[idx]==',')
        new_heap_node = make_new_Node(0,',',NULL,NULL,0);
    else if(encoded_tree[idx]==')')
    {
        new_heap_node = make_new_Node(0,' ',decode_heap[decode_heap_size-2],decode_heap[decode_heap_size],0);
        decode_heap_size-=4;
    }
    else
        new_heap_node = make_new_Node(1,encoded_tree[idx],NULL,NULL,0);
    decode_Heap_push(new_heap_node);
    return;
}
```

From this tree, I write binary code of each alphabet character.



```
    for(int i=0;i<ALPHA_NUM;i++)
        if(alpha_code_len[i]!=0)
        {
            fputc(AL_a+i,fp_output);
            fputs(" :",fp_output);
            decode_mask_out(alpha_code_len[i],alpha_code_num[i],fp_output);
            fputs("\n",fp_output);
        }
```

Using tree, I decode hw3_input.txt file.



```
    while((c=fgetc(fp_input))!=EOF)
    {
        if(start==1)
            node = rootNode;

        if(c=='0')
        {
            // printf("0");
            node = node->child_0;
        }
        else
        {
            // printf("1");
            node = node->child_1;
        }

        if(node->is_leaf==1)
        {
            fputc(node->alpha,fp_output);
            start=1;
        }
        else
        {
            start=0;
        }
    }
```

Thank you for reading.