**Exercise 7. Answer Sheet**

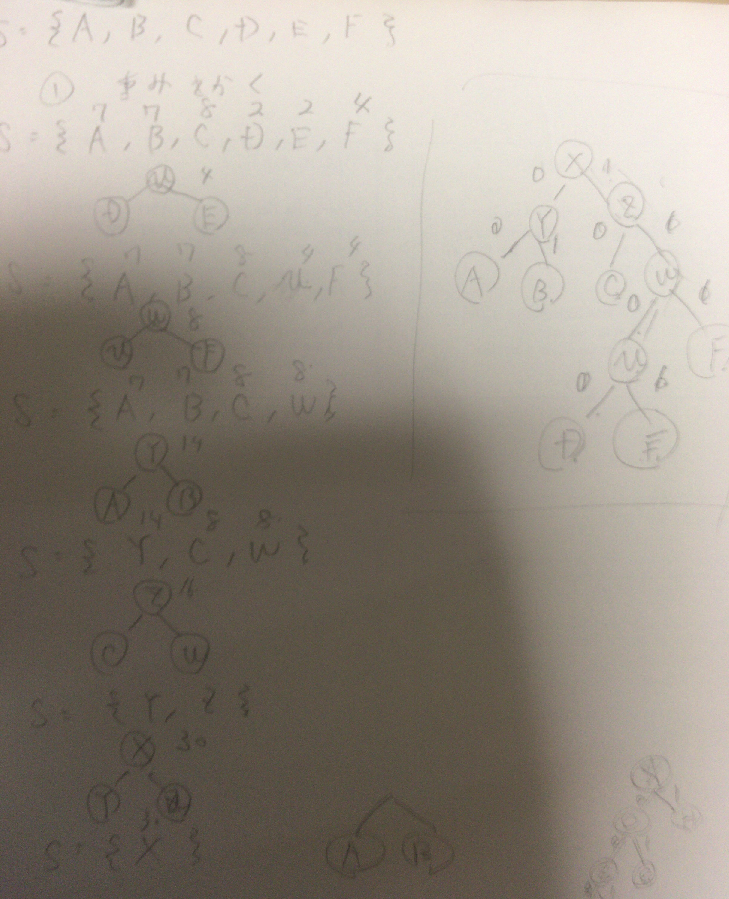
Student's Name: Tomonori Masubuci Student's ID: s1240078

***Problem 1.***  (20 point) Consider following sequence of letters:

ABBCACCEACBCCFABCDAFEABFFADBBC

a) (10 points) Construct Huffman encoding tree for the above sequence and show it below

Put your answer here



b) (10 points) What is the code for each letter:

A: 00

B: 01

C: 10

D: 1100

E: 1101

F: 111

***Problem 2.*** (25 points) What is the Huffman code for the following set of frequencies, based on the first 8 Fibonacchi numbers?

A: 1, B: 1, C: 2, D: 3, E: 5, F: 8, G: 13, H: 21

A: 0000000

B: 0000001

C: 000001

D: 00001

E: 0001

F: 001

G: 01

H: 1

***Problem 3.*** (15 points)Write your name in English letters and construct Huffman tree and code for it. Show your tree and code below.

tomonori masubuci

a: 1

b: 1

c: 1

n: 1

r: 1

s: 1

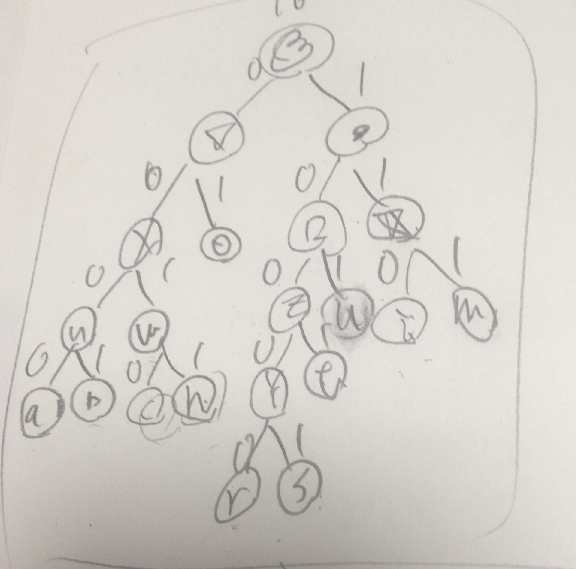
t: 1

i: 2

m: 2

u: 2

o: 3



a: 0000 b: 0001 c: 0010 n: 0011 r: 10000 s: 10001 t: 1001 u: 101 i: 110 m: 111

***Problem 4.*** (40 points)Make a program implementing Huffman encoding. Upload your code with usage example.

#include <stdio.h>

#include <stdlib.h>

#define MAXTREE 100

struct MinHeapNode

{

char data;

unsigned freq;

struct MinHeapNode \*left, \*right;

};

struct MinHeap

{

unsigned size;

unsigned capacity;

struct MinHeapNode\*\* array;

};

struct MinHeapNode\* newNode(char data, unsigned freq){

struct MinHeapNode\* temp= (struct MinHeapNode\*)malloc

(sizeof(struct MinHeapNode));

temp->left = temp->right = NULL;

temp->data = data;

temp->freq = freq;

return temp;

}

struct MinHeap\* createMinHeap(unsigned capacity){

struct MinHeap\* minHeap= (struct MinHeap\*)malloc(sizeof(struct MinHeap));

minHeap->size = 0;

minHeap->capacity = capacity;

minHeap->array= (struct MinHeapNode\*\*)malloc(minHeap->

capacity \* sizeof(struct MinHeapNode\*));

return minHeap;

}

void swapMinHeapNode(struct MinHeapNode\*\* a, struct MinHeapNode\*\* b);

void minHeapify(struct MinHeap\* minHeap, int idx);

int isSizeOne(struct MinHeap\* minHeap);

struct MinHeapNode\* extractMin(struct MinHeap\* minHeap)

{

struct MinHeapNode\* temp = minHeap->array[0];

minHeap->array[0] = minHeap->array[minHeap->size - 1];

--minHeap->size;

minHeapify(minHeap, 0);

return temp;

}

void insertMinHeap(struct MinHeap\* minHeap,struct MinHeapNode\* minHeapNode);

void buildMinHeap(struct MinHeap\* minHeap);

void print\_arr(int arr[], int n);

int isLeaf(struct MinHeapNode\* root);

void cprint(struct MinHeapNode\* root, int arr[], int top);

void doHuffman(char data[], int freq[], int size);

struct MinHeap\* createAndBuildMinHeap(char data[], int freq[], int size){

struct MinHeap\* minHeap = createMinHeap(size);

for (int i = 0; i < size; ++i)

minHeap->array[i] = newNode(data[i], freq[i]);

minHeap->size = size;

buildMinHeap(minHeap);

return minHeap;

}

struct MinHeapNode\* buildHuffmanTree(char data[], int freq[], int size){

struct MinHeapNode \*left, \*right, \*top;

struct MinHeap\* minHeap = createAndBuildMinHeap(data, freq, size);

while (!isSizeOne(minHeap)) {

left = extractMin(minHeap);

right = extractMin(minHeap);

top = newNode('$', left->freq + right->freq);

top->left = left;

top->right = right;

insertMinHeap(minHeap, top);

}

return extractMin(minHeap);

}

void cprint(struct MinHeapNode\* root, int arr[], int top);

void doHuffman(char data[], int freq[], int size);

int main(){

char arr[] = { 'A', 'B', 'C', 'D','E','F','G','H'};

int freq[] = { 1, 1, 2, 3, 5, 8, 13, 21};

int size = sizeof(arr) / sizeof(arr[0]);

doHuffman(arr, freq, size);

return 0;

}

void swapMinHeapNode(struct MinHeapNode\*\* a, struct MinHeapNode\*\* b)

{

struct MinHeapNode\* t = \*a;

\*a = \*b;

\*b = t;

}

void minHeapify(struct MinHeap\* minHeap, int idx)

{

int smallest = idx;

int left = 2 \* idx + 1;

int right = 2 \* idx + 2;

if (left < minHeap->size && minHeap->array[left]->freq < minHeap->array[smallest]->freq)

smallest = left;

if (right < minHeap->size && minHeap->array[right]->freq < minHeap->array[smallest]->freq) smallest = right;

if (smallest != idx) {

swapMinHeapNode(&minHeap->array[smallest],&minHeap->array[idx]);

minHeapify(minHeap, smallest);

}

}

int isSizeOne(struct MinHeap\* minHeap)

{

return (minHeap->size == 1);

}

void doHuffman(char data[], int freq[], int size){

struct MinHeapNode\* root = buildHuffmanTree(data, freq, size);

int arr[MAXTREE], top = 0;

cprint(root, arr, top);

}

void insertMinHeap(struct MinHeap\* minHeap,struct MinHeapNode\* minHeapNode)

{

++minHeap->size;

int i = minHeap->size - 1;

while (i && minHeapNode->freq < minHeap->array[(i - 1) / 2]->freq){

minHeap->array[i] = minHeap->array[(i - 1) / 2];

i = (i - 1) / 2;

}

minHeap->array[i] = minHeapNode;

}

void buildMinHeap(struct MinHeap\* minHeap)

{

int i;

int n = minHeap->size - 1;

for (i = (n - 1) / 2;i >= 0; --i)

minHeapify(minHeap, i);

}

void print\_arr(int arr[], int n){

int i;

for (i = 0;i < n;++ i)

printf("%d", arr[i]);

printf("\n");

}

int isLeaf(struct MinHeapNode\* root){

return !(root->left) && !(root->right);

}

void cprint(struct MinHeapNode\* root, int arr[], int top){

if (root->left)

{

arr[top] = 0;

cprint(root->left, arr, top + 1);

}

if (root->right)

{

arr[top] = 1;

cprint(root->right, arr, top + 1);

}

if (isLeaf(root))

{

printf("%c: ", root->data);

print\_arr(arr, top);

}

}

H: 0

G: 10

F: 110

E: 1110

D: 11110

C: 111110

A: 1111110

B: 1111111