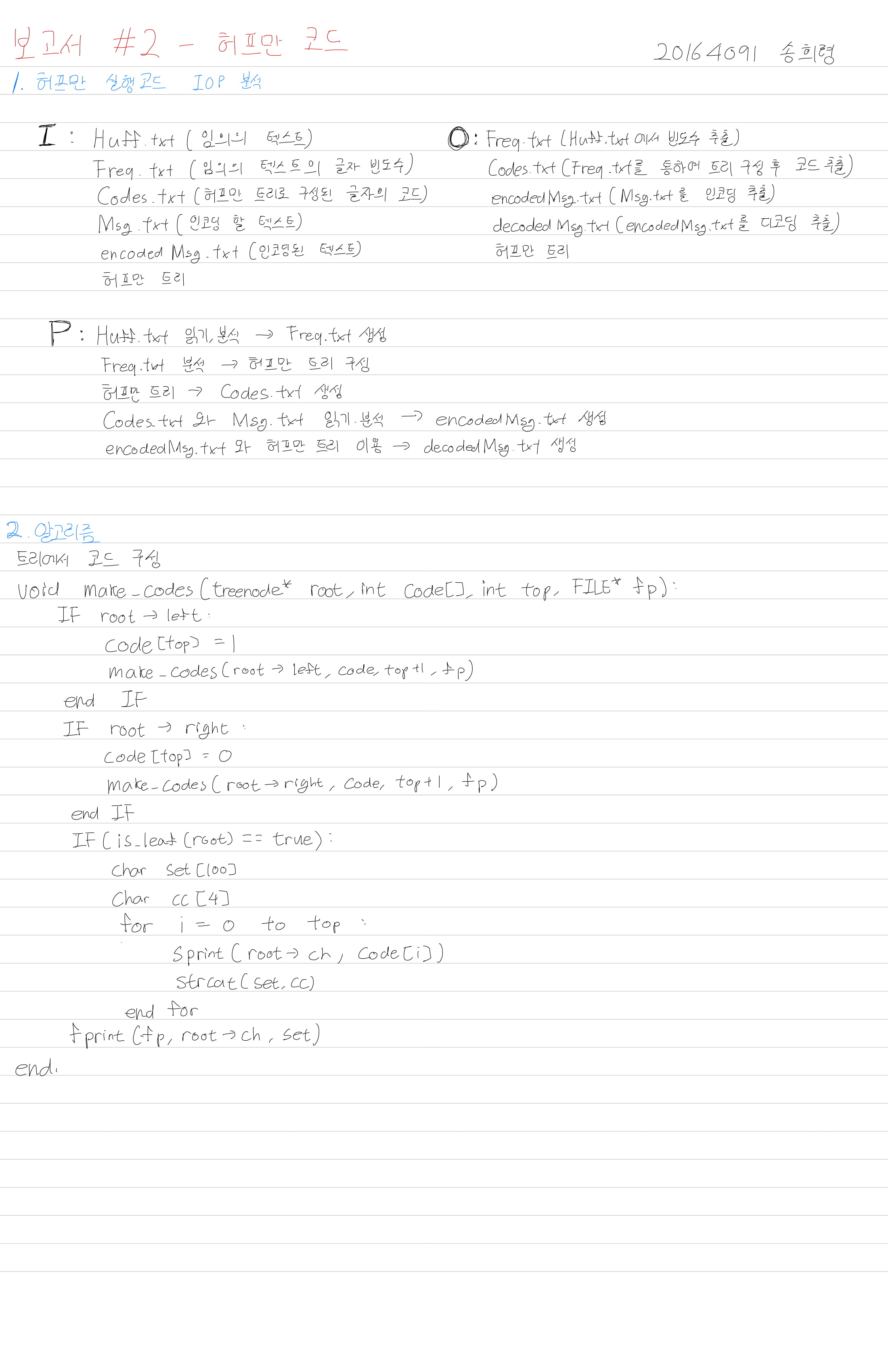
데이터구조 2

보고서 #2

(허프만 코드)

20164091

송희령



3. 코딩

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct treenode {

int weight;

char ch;

struct treenode\* left;

struct treenode\* right;

}treenode;

typedef struct element {

treenode\* ptree;

char ch;

int key;

}element;

typedef struct heaptype{

element heap[100];

int heap\_size;

}heaptype;

코드 시작 부분의 구조체 정의 입니다. 트리노드, 원소와 히프타입의 정의입니다.

heaptype\* create() {

return (heaptype\*)malloc(sizeof(heaptype));

}

void init(heaptype\* h) {

h->heap\_size = 0;

}

히프의 생성과 초기화를 나타내는 함수입니다.

treenode\* make\_tree(treenode\* left, treenode\* right) {

treenode\* node = (treenode\*)malloc(sizeof(treenode));

node->left = left;

node->right = right;

return node;

}

void destroy\_tree(treenode\* root) {

if (root == NULL)

return;

destroy\_tree(root->left);

destroy\_tree(root->right);

free(root);

}

int is\_leaf(treenode\* root) {

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

}

트리의 생성과 삭제를 나타내는 함수와 리프노드의 유무를 반환하는 함수입니다.

element delete\_min\_heap(heaptype\* h) {

int parent = 1, child = 2;

element item, temp;

item = h->heap[1];

temp = h->heap[(h->heap\_size)--];

while (child <= h->heap\_size) {

if ((child < h->heap\_size) && h->heap[child].key > h->heap[child + 1].key)

child++;

if (temp.key < h->heap[child].key)

break;

h->heap[parent] = h->heap[child];

parent = child;

child += 2;

}

h->heap[parent] = temp;

return item;

}

void insert\_min\_heap(heaptype\* h, element item) {

int i;

i = ++(h->heap\_size);

while ((i != 1) && (item.key < h->heap[i / 2].key)) {

h->heap[i] = h->heap[i / 2];

i /= 2;

}

h->heap[i] = item;

}

트리의 삽입과 삭제를 나타내는 함수입니다.

void make\_codes(treenode\* root, int code[], int top, FILE\* fp) {

if (root->left) {

code[top] = 1;

make\_codes(root->left, code, top + 1, fp);

}

if (root->right) {

code[top] = 0;

make\_codes(root->right, code, top + 1, fp);

}

if (is\_leaf(root)) {

char set[100] = { 0 };

char cc[4];

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

for (int i = 0; i < top; i++) {

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

sprintf(cc, "%d", code[i]);

strcat(set, cc);

}

printf("\n");

fprintf(fp, "%c %s\n", root->ch, set);

}

}

void preOrder(treenode\* root) {//전위탐색

if (root != NULL) {

printf("%d\n", root->weight);

preOrder(root->left);

preOrder(root->right);

}

}

허프만 코드 생성을 위한 함수와 전위탐색 함수 입니다.

이상 허프만 코드는 책 위주로 참고하여 작성하였습니다.

int main() {

int fre[30] = { 0 };//빈도수 구분

char ch;//글자 읽기

FILE\* fp\_huff = fopen("huff.txt", "r");//huff.txt파일 읽기

printf("------파일 내용-----\n");

while (1) {//huff.txt파일 출력

char tmp = fgetc(fp\_huff);

printf("%c", tmp);

if (tmp == EOF)

break;

}

fseek(fp\_huff, 0, SEEK\_SET);//huff파일 포인터 초기화

printf("\n\n------각 글자의 횟수-----\n");

do {//EOF가 나올때까지 파일의 글자 읽기

ch = fgetc(fp\_huff);

if (ch == ' ') {

fre[26]++;//공백 26번자리 증가

continue;

}

if (ch == '\n') {

fre[27]++;//줄바꿈 27번자리 증가

continue;

}

if (ch == '.') {

fre[28]++;//마침표 28번자리 증가

continue;

}

if (ch == EOF) {

fre[29]++;//EOF 29번자리 증가

continue;

}

fre[ch - 'a']++;//a부터 z까지 ASCII를 이용하여 0~25번에 각각 할당하여 빈도수 증가

} while (ch != EOF);//EOF가 나올때까지 파일의 글자 읽기

fclose(fp\_huff);//huff.txt 종료

FILE\* fp\_freq = fopen("freq.txt", "w+");//freq.txt 쓰기생성

int line = 0;//텍스트파일 라인수 체크

for (int i = 0; i < 30; i++) {//전체 허용 글자수만큼 반복, 글자수 표현

if (fre[i]) {//해당 글자가 1번이상 등장하면 실행, 글자반복수 출력

line++;//텍스트파일 라인수 증가

if (i == 26) {

printf("공백(-)의 횟수 : %d\n", fre[i]);//cmd화면출력

fprintf(fp\_freq, "- %d\n", fre[i]);//화일출력

continue;

}

if (i == 27) {

printf("새줄(!)의 횟수 : %d\n", fre[i]);

fprintf(fp\_freq, "! %d\n", fre[i]);

continue;

}

if (i == 28) {

printf("마침표(.)의 횟수 : %d\n", fre[i]);

fprintf(fp\_freq, ". %d\n", fre[i]);

continue;

}

if (i == 29) {

printf("EOF(+)의 횟수 : %d\n", fre[i]);

fprintf(fp\_freq, "+ %d", fre[i]);

continue;

}

printf("%c의 횟수 : %d\n", 'a' + i, fre[i]);

fprintf(fp\_freq, "%c %d\n", i + 'a', fre[i]);

}

}

fseek(fp\_freq, 0, SEEK\_SET);//freq파일 포인터 초기화

char alp2[30];//freq를 불러서 읽을 알파벳 배열

int fre2[30];//freq를 불러서 읽을 빈도수 배열

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

fscanf(fp\_freq, "%c %d\n", &alp2[i], &fre2[i]);//파일 읽기. 알파벳,빈도수

메인 함수 의 단계 1인 문자 빈도수 산출(freq.txt) 입니다.

treenode\* node, \*x;

heaptype\* heap;

heap = create();

init(heap);

element e, e1, e2;

//노드생성

for (int i = 0; i < line; i++) {

node = make\_tree(NULL, NULL);

e.ch = node->ch = alp2[i];

e.key = node->weight = fre2[i];

e.ptree = node;

insert\_min\_heap(heap, e);//최소정렬로 정렬하여 삽입.

}

printf("\n----트리 생성-----\n");

for (int i = 1; i < line; i++) {

e1 = delete\_min\_heap(heap);

e2 = delete\_min\_heap(heap);

x = make\_tree(e1.ptree, e2.ptree);

e.key = x->weight = e1.key + e2.key;

e.ptree = x;

printf("%d + %d -> %d\n", e1.key, e2.key, e.key);

insert\_min\_heap(heap, e);

}

printf("\n----트리 전위 탐색-----\n");

preOrder(e.ptree);

메인 함수의 단계 2인 허프만 트리(huff-tree) 구성입니다.

int code[100], top = 0;

element huffmancode;

FILE\* fp\_codes = fopen("codes.txt", "w+");//codes.txt 쓰기생성

huffmancode = delete\_min\_heap(heap);

printf("\n----허프만 코드 출력-----\n");

make\_codes(huffmancode.ptree, code, top, fp\_codes);//허프만코드 출력

fseek(fp\_codes, 0, SEEK\_SET);//huff파일 포인터 초기화

메인 함수의 단계 3인 허프만 코드 생성(codes.txt) 입니다.

char hword[30], hcode[30][100];//허프만 코드를 배열에 저장

for (int i = 0; i < line; i++) {

fscanf(fp\_codes, "%c %s\n", &hword[i], hcode[i]);

}

//허프만 인코딩

FILE\* fp\_msg = fopen("msg.txt", "r");//메세지파일 읽기

FILE\* fp\_encodedMsg = fopen("encodedMsg.txt", "w+");//인코딩파일 쓰기

char a, compare, encmsg[10000] = { 0 };//txt파일의 글자 받아오기, 비교대상, 인코딩된메세지

do {

a = fgetc(fp\_msg);

if (a == ' ') {

compare = '-';

for (int i = 0; i < line; i++) {

if (hword[i] == compare)

strcat(encmsg, hcode[i]);

}

continue;

}

if (a == '\n') {

compare = '!';

for (int i = 0; i < line; i++) {

if (hword[i] == compare)

strcat(encmsg, hcode[i]);

}

continue;

}

if (a == EOF) {

compare = '+';

for (int i = 0; i < line; i++) {

if (hword[i] == compare)

strcat(encmsg, hcode[i]);

}

continue;

}

compare = a;

for (int i = 0; i < line; i++) {

if (hword[i] == compare) {

strcat(encmsg, hcode[i]);

continue;

}

}

} while (a != EOF);

fprintf(fp\_encodedMsg, "%s", encmsg);

printf("\n-----msg.txt 인코딩 결과-----\n%s\n\n%s\n",encmsg,"-----디코딩 문장-----");

메인 함수의 단계 4인 인코딩 단계(encodedMsg.txt)입니다.

//허프만 디코딩

fseek(fp\_encodedMsg, 0, SEEK\_SET);

char before\_decmsg[10000] = { 0 };

fscanf(fp\_encodedMsg, "%s", &before\_decmsg);

treenode\* plot = e.ptree;

fseek(fp\_encodedMsg, 0, SEEK\_SET);

for (int i = 0; i < strlen(before\_decmsg) + 1; i++) {

char codeone = fgetc(fp\_encodedMsg);

if (is\_leaf(plot)) {

printf("%c", plot->ch);

plot = e.ptree;

}

if (codeone == '0')

plot = plot->right;

else if (codeone == '1')

plot = plot->left;

}

메인 함수의 단계 5인 디코딩 단계(decoded.txt)입니다.

//특수기호 제외

printf("\n\n-----특수기호 제외-----\n");

plot = e.ptree;

fseek(fp\_encodedMsg, 0, SEEK\_SET);

for (int i = 0; i < strlen(before\_decmsg) + 1; i++) {

char codeone = fgetc(fp\_encodedMsg);

if (is\_leaf(plot)) {

if (plot->ch == '-')

printf(" ");

else if (plot->ch == '!')

printf("\n");

else if (plot->ch == '+')

printf("끝");

else

printf("%c", plot->ch);

plot = e.ptree;

}

if (codeone == '0')

plot = plot->right;

else if (codeone == '1')

plot = plot->left;

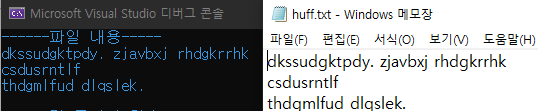
}

메인 함수의 단계 6인 결과 출력 입니다.

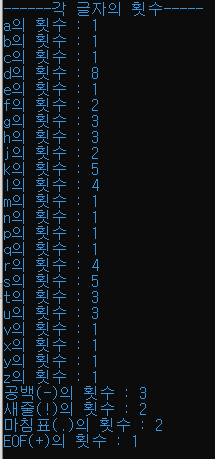
실행 검증을 위한 전체적인 코드는 레포트 후반부에 기술하겠습니다.

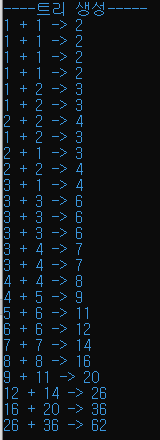
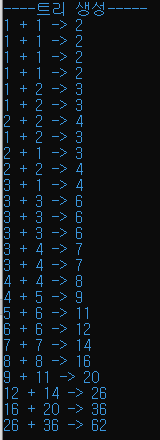
4. 테스트

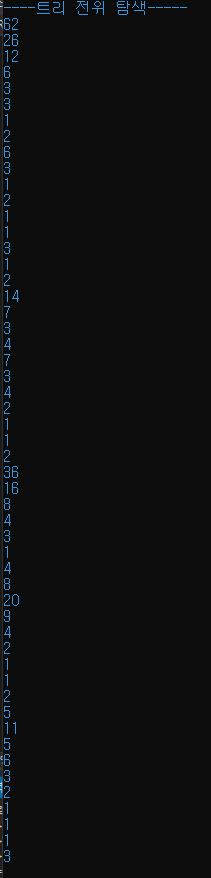
Huff.txt 불러오기

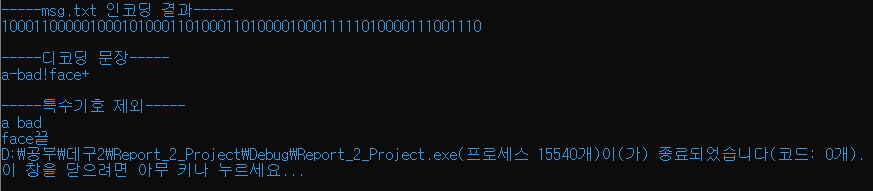


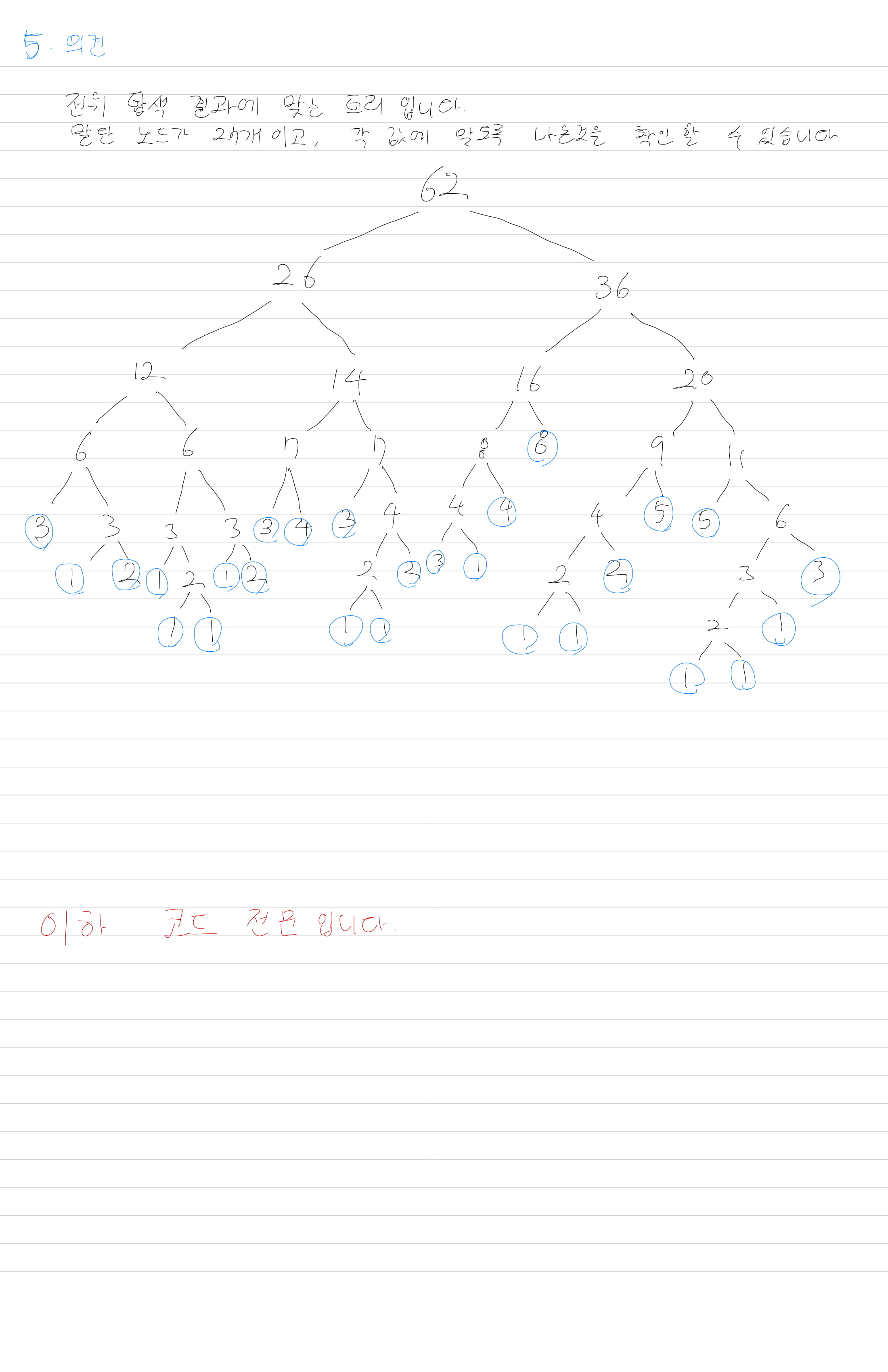
각 글자의 횟수(freq)



글자 횟수 트리 생성 트리 전위 순회



실행 결과



검증 코드

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

typedef struct treenode {

int weight;

char ch;

struct treenode\* left;

struct treenode\* right;

}treenode;

typedef struct element {

treenode\* ptree;

char ch;

int key;

}element;

typedef struct heaptype{

element heap[100];

int heap\_size;

}heaptype;

heaptype\* create() {

return (heaptype\*)malloc(sizeof(heaptype));

}

void init(heaptype\* h) {

h->heap\_size = 0;

}

treenode\* make\_tree(treenode\* left, treenode\* right) {

treenode\* node = (treenode\*)malloc(sizeof(treenode));

node->left = left;

node->right = right;

return node;

}

void destroy\_tree(treenode\* root) {

if (root == NULL)

return;

destroy\_tree(root->left);

destroy\_tree(root->right);

free(root);

}

int is\_leaf(treenode\* root) {

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

}

element delete\_min\_heap(heaptype\* h) {

int parent = 1, child = 2;

element item, temp;

item = h->heap[1];

temp = h->heap[(h->heap\_size)--];

while (child <= h->heap\_size) {

if ((child < h->heap\_size) && h->heap[child].key > h->heap[child + 1].key)

child++;

if (temp.key < h->heap[child].key)

break;

h->heap[parent] = h->heap[child];

parent = child;

child += 2;

}

h->heap[parent] = temp;

return item;

}

void insert\_min\_heap(heaptype\* h, element item) {

int i;

i = ++(h->heap\_size);

while ((i != 1) && (item.key < h->heap[i / 2].key)) {

h->heap[i] = h->heap[i / 2];

i /= 2;

}

h->heap[i] = item;

}

void make\_codes(treenode\* root, int code[], int top, FILE\* fp) {

if (root->left) {

code[top] = 1;

make\_codes(root->left, code, top + 1, fp);

}

if (root->right) {

code[top] = 0;

make\_codes(root->right, code, top + 1, fp);

}

if (is\_leaf(root)) {

char set[100] = { 0 };

char cc[4];

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

for (int i = 0; i < top; i++) {

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

sprintf(cc, "%d", code[i]);

strcat(set, cc);

}

printf("\n");

fprintf(fp, "%c %s\n", root->ch, set);

}

}

void preOrder(treenode\* root) {//전위탐색

if (root != NULL) {

printf("%d\n", root->weight);

preOrder(root->left);

preOrder(root->right);

}

}

int main() {

int fre[30] = { 0 };//빈도수 구분

char ch;//글자 읽기

FILE\* fp\_huff = fopen("huff.txt", "r");//huff.txt파일 읽기

printf("------파일 내용-----\n");

while (1) {//huff.txt파일 출력

char tmp = fgetc(fp\_huff);

printf("%c", tmp);

if (tmp == EOF)

break;

}

fseek(fp\_huff, 0, SEEK\_SET);//huff파일 포인터 초기화

printf("\n\n------각 글자의 횟수-----\n");

do {//EOF가 나올때까지 파일의 글자 읽기

ch = fgetc(fp\_huff);

if (ch == ' ') {

fre[26]++;//공백 26번자리 증가

continue;

}

if (ch == '\n') {

fre[27]++;//줄바꿈 27번자리 증가

continue;

}

if (ch == '.') {

fre[28]++;//마침표 28번자리 증가

continue;

}

if (ch == EOF) {

fre[29]++;//EOF 29번자리 증가

continue;

}

fre[ch - 'a']++;//a부터 z까지 ASCII를 이용하여 0~25번에 각각 할당하여 빈도수 증가

} while (ch != EOF);//EOF가 나올때까지 파일의 글자 읽기

fclose(fp\_huff);//huff.txt 종료

FILE\* fp\_freq = fopen("freq.txt", "w+");//freq.txt 쓰기생성

int line = 0;//텍스트파일 라인수 체크

for (int i = 0; i < 30; i++) {//전체 허용 글자수만큼 반복, 글자수 표현

if (fre[i]) {//해당 글자가 1번이상 등장하면 실행, 글자반복수 출력

line++;//텍스트파일 라인수 증가

if (i == 26) {

printf("공백(-)의 횟수 : %d\n", fre[i]);//cmd화면출력

fprintf(fp\_freq, "- %d\n", fre[i]);//화일출력

continue;

}

if (i == 27) {

printf("새줄(!)의 횟수 : %d\n", fre[i]);

fprintf(fp\_freq, "! %d\n", fre[i]);

continue;

}

if (i == 28) {

printf("마침표(.)의 횟수 : %d\n", fre[i]);

fprintf(fp\_freq, ". %d\n", fre[i]);

continue;

}

if (i == 29) {

printf("EOF(+)의 횟수 : %d\n", fre[i]);

fprintf(fp\_freq, "+ %d", fre[i]);

continue;

}

printf("%c의 횟수 : %d\n", 'a' + i, fre[i]);

fprintf(fp\_freq, "%c %d\n", i + 'a', fre[i]);

}

}

fseek(fp\_freq, 0, SEEK\_SET);//freq파일 포인터 초기화

char alp2[30];//freq를 불러서 읽을 알파벳 배열

int fre2[30];//freq를 불러서 읽을 빈도수 배열

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

fscanf(fp\_freq, "%c %d\n", &alp2[i], &fre2[i]);//파일 읽기. 알파벳,빈도수

treenode\* node, \*x;

heaptype\* heap;

heap = create();

init(heap);

element e, e1, e2;

//노드생성

for (int i = 0; i < line; i++) {

node = make\_tree(NULL, NULL);

e.ch = node->ch = alp2[i];

e.key = node->weight = fre2[i];

e.ptree = node;

insert\_min\_heap(heap, e);//최소정렬로 정렬하여 삽입.

}

printf("\n----트리 생성-----\n");

for (int i = 1; i < line; i++) {

e1 = delete\_min\_heap(heap);

e2 = delete\_min\_heap(heap);

x = make\_tree(e1.ptree, e2.ptree);

e.key = x->weight = e1.key + e2.key;

e.ptree = x;

printf("%d + %d -> %d\n", e1.key, e2.key, e.key);

insert\_min\_heap(heap, e);

}

printf("\n----트리 전위 탐색-----\n");

preOrder(e.ptree);

int code[100], top = 0;

element huffmancode;

FILE\* fp\_codes = fopen("codes.txt", "w+");//codes.txt 쓰기생성

huffmancode = delete\_min\_heap(heap);

printf("\n----허프만 코드 출력-----\n");

make\_codes(huffmancode.ptree, code, top, fp\_codes);//허프만코드 출력

fseek(fp\_codes, 0, SEEK\_SET);//huff파일 포인터 초기화

char hword[30], hcode[30][100];//허프만 코드를 배열에 저장

for (int i = 0; i < line; i++) {

fscanf(fp\_codes, "%c %s\n", &hword[i], hcode[i]);

}

//허프만 인코딩

FILE\* fp\_msg = fopen("msg.txt", "r");//메세지파일 읽기

FILE\* fp\_encodedMsg = fopen("encodedMsg.txt", "w+");//인코딩파일 쓰기

char a, compare, encmsg[10000] = { 0 };//txt파일의 글자 받아오기, 비교대상, 인코딩된메세지

do {

a = fgetc(fp\_msg);

if (a == ' ') {

compare = '-';

for (int i = 0; i < line; i++) {

if (hword[i] == compare)

strcat(encmsg, hcode[i]);

}

continue;

}

if (a == '\n') {

compare = '!';

for (int i = 0; i < line; i++) {

if (hword[i] == compare)

strcat(encmsg, hcode[i]);

}

continue;

}

if (a == EOF) {

compare = '+';

for (int i = 0; i < line; i++) {

if (hword[i] == compare)

strcat(encmsg, hcode[i]);

}

continue;

}

compare = a;

for (int i = 0; i < line; i++) {

if (hword[i] == compare) {

strcat(encmsg, hcode[i]);

continue;

}

}

} while (a != EOF);

fprintf(fp\_encodedMsg, "%s", encmsg);

printf("\n-----msg.txt 인코딩 결과-----\n%s\n\n%s\n",encmsg,"-----디코딩 문장-----");

//허프만 디코딩

fseek(fp\_encodedMsg, 0, SEEK\_SET);

char before\_decmsg[10000] = { 0 };

fscanf(fp\_encodedMsg, "%s", &before\_decmsg);

treenode\* plot = e.ptree;

fseek(fp\_encodedMsg, 0, SEEK\_SET);

for (int i = 0; i < strlen(before\_decmsg) + 1; i++) {

char codeone = fgetc(fp\_encodedMsg);

if (is\_leaf(plot)) {

printf("%c", plot->ch);

plot = e.ptree;

}

if (codeone == '0')

plot = plot->right;

else if (codeone == '1')

plot = plot->left;

}

//특수기호 제외

printf("\n\n-----특수기호 제외-----\n");

plot = e.ptree;

fseek(fp\_encodedMsg, 0, SEEK\_SET);

for (int i = 0; i < strlen(before\_decmsg) + 1; i++) {

char codeone = fgetc(fp\_encodedMsg);

if (is\_leaf(plot)) {

if (plot->ch == '-')

printf(" ");

else if (plot->ch == '!')

printf("\n");

else if (plot->ch == '+')

printf("끝");

else

printf("%c", plot->ch);

plot = e.ptree;

}

if (codeone == '0')

plot = plot->right;

else if (codeone == '1')

plot = plot->left;

}

fclose(fp\_codes);

fclose(fp\_encodedMsg);

fclose(fp\_freq);

fclose(fp\_huff);

fclose(fp\_msg);

return 0;

}