알고리즘 과제

Practice.11

**학번 : 201402432**

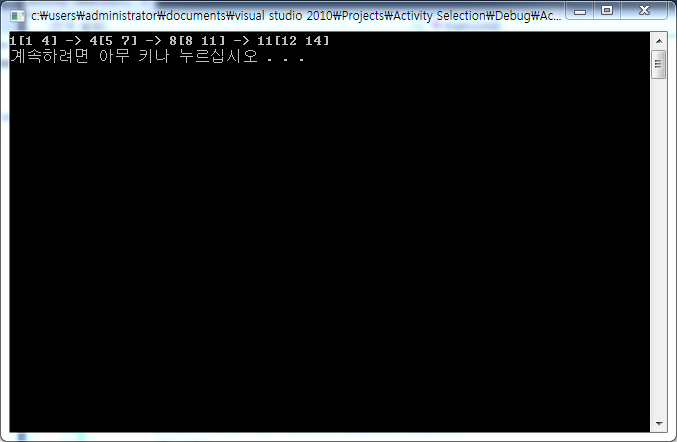
**이름 : 조디모데**

**11-1 Activity Selection using Greedy Algorithm**

* 알고리즘 설명

1. 배열 S중 선택하지 않은 것 중 가장 작은 값을 선택한다.
2. 가장 작은 S값과 겹치는 값을 S에서 모두 제외시킨다.
3. S가 비어있을 때 까지 반복한다.

* 컴파일 방법  
  Text파일을 바탕화면에 넣는다  
  "C:\\Users\\Administrator\\Desktop\\sample\_selection.txt
* 프로그램 결과값



* Code ( .C )

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#define LEN 11

int S[LEN] ;

int F[LEN] ;

void getArr(){

FILE \*fps ;

int i, n1, n2 ;

fps = fopen("C:\\Users\\Administrator\\Desktop\\sample\_selection.txt","rt");

for(i=0 ; i < LEN ; i++){

fscanf(fps,"%d %d",&n1, &n2) ;

S[i] = n1 ;

F[i] = n2 ;

}

fclose(fps) ;

}

int Iterative\_greedy\_algorithm(int \*A, int k){

int i ;

k=0 ;

A[k] = k ;

for(i=0 ; i<LEN ; i++){

if(S[i] >= F[A[k]])

A[++k] = i ;

}

return k ;

}

int main(void){

int A[LEN] ;

int i, k=0 ;

getArr() ;

k = Iterative\_greedy\_algorithm(A, k) ;

for(i=0 ; i<=k ; i++){

printf("%d[%d %d]",A[i]+1, S[A[i]], F[A[i]]) ;

if(i==k)

printf("\n");

else

printf(" -> ") ;

}

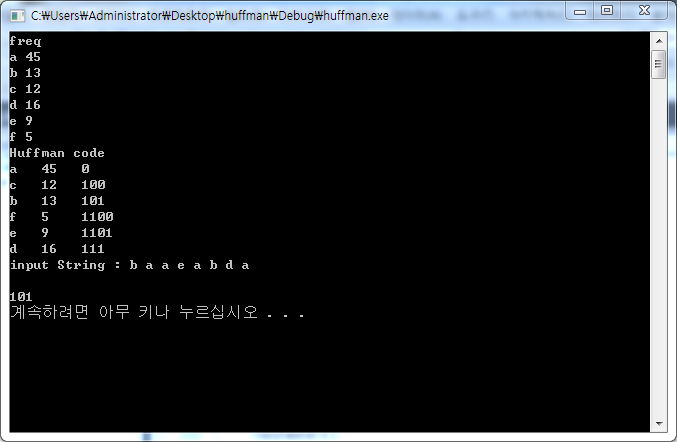
system("pause") ;

return 0;

}

**11-2 Huffman Code**

* 알고리즘 설명  
  문자들의 발생 빈도수에 따라 가변 길이 코드를 생성, 고정 길이로 사용할 때보다 데이터 양이 줄어든다는 섀넌-파노 원리를 이용한 압축 방식  
  1) 초기화 : 모든 기호를 출현 빈도수에 따라 나열함  
  2) 축소 : 발생 확률이 가장 낮은 두 개의 심볼을 결합 시킴  
   결합된 노드와 나머지 낮은 심볼을 결합  
   두 개의 심볼이 남을 때 까지 반복  
  3) 확장 : 최종 두 개의 심볼에 ‘0’, ‘1’을 할당  
   축소 과정의 반대 방향으로 확장하면서 ‘0’, ‘1’을 할당  
   소스 심볼의 개수까지 반복
* 컴파일 방법  
  TEXT 를 바탕화면에 넣는다  
  "C:\\Users\\Administrator\\Desktop\\sample\_huffman.txt"
* 프로그램 결과



Code ( .cpp )

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define ALPHABET 26

#define MAX\_LEN 255

#define MAX\_ELEMENT 1000

typedef struct{

char alpha;

int freq;

}AlphaType;

typedef struct TreeNode{

AlphaType weight;

TreeNode \*left\_child;

TreeNode \*right\_child;

}TreeNode;

typedef struct{

TreeNode \*pTree;

int key;

}Element;

typedef struct{

Element heap[MAX\_ELEMENT];

int heap\_size;

}HeapType;

void InsertHeap(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;

}

Element DeleteHeap(HeapType \*h){

int parent=1,child=2;

Element data,temp;

data = h->heap[parent];

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 data;

}

void inputmake(AlphaType \*A){

FILE \*fps;

char ap=NULL;

int freq=0;

int i;

int temp=0;

fps = fopen("C:\\Users\\Administrator\\Desktop\\sample\_huffman.txt","rt");

for(i=0; i<6; i++){

fscanf(fps,"%c %d %d",&ap,&freq,&temp);

A[i].alpha = ap;

A[i].freq = freq;

}

fclose(fps);

}

TreeNode\* MakeNode(TreeNode \*left,TreeNode \*right){

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

node->left\_child=left;

node->right\_child=right;

return node;

}

void PrintTree(TreeNode \*p,int i,char \*pCode){

if(p)

{

i++;

pCode[i]='0';

PrintTree(p->left\_child,i,pCode);

pCode[i]='1';

PrintTree(p->right\_child,i,pCode);

pCode[i]='\0';

if(p->left\_child == NULL && p->right\_child == NULL)

{

printf("%c %d\t %s\t\n",p->weight.alpha,p->weight.freq,pCode);

}

}

}

void HuffmanTree(AlphaType \*pArr,int n){

TreeNode \*node,\*temp;

Element e,e1,e2;

HeapType heap;

char binaryCode[100];

int i;

heap.heap\_size=0;

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

{

node=MakeNode(NULL,NULL);

node->weight.alpha=pArr[i].alpha;

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

e.pTree=node;

InsertHeap(&heap,e);

}

for(i=0;i<n-1;i++){

e1=DeleteHeap(&heap);

e2=DeleteHeap(&heap);

temp=MakeNode(e1.pTree,e2.pTree);

e.key=temp->weight.freq=e1.key+e2.key;

e.pTree=temp;

InsertHeap(&heap,e);

}

e = DeleteHeap(&heap);

PrintTree(e.pTree,-1,binaryCode);

}

void Printcode(TreeNode \*p,int i,char \*pCode,char alpha){

if(p)

{

i++;

pCode[i]='0';

Printcode(p->left\_child,i,pCode,alpha);

pCode[i]='1';

Printcode(p->right\_child,i,pCode,alpha);

pCode[i]='\0';

if(p->left\_child == NULL && p->right\_child == NULL && p->weight.alpha == alpha )

{

printf("%s",pCode);

}

}

}

void HuffmanTreeC(AlphaType \*pArr,int n,char alpha){

TreeNode \*node,\*temp;

Element e,e1,e2;

HeapType heap;

char binaryCode[100];

int i;

heap.heap\_size=0;

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

{

node=MakeNode(NULL,NULL);

node->weight.alpha=pArr[i].alpha;

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

e.pTree=node;

InsertHeap(&heap,e);

}

for(i=0;i<n-1;i++){

e1=DeleteHeap(&heap);

e2=DeleteHeap(&heap);

temp=MakeNode(e1.pTree,e2.pTree);

e.key=temp->weight.freq=e1.key+e2.key;

e.pTree=temp;

InsertHeap(&heap,e);

}

e = DeleteHeap(&heap);

Printcode(e.pTree,-1,binaryCode,alpha);

}

void Init(AlphaType \*p){

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

p[i].alpha=i+65;

p[i].freq=0;

}

}

int main(void){

int i;

AlphaType \*A;

char \*string = (char\*)malloc(sizeof(char)\*MAX\_LEN);

A = (AlphaType\*)malloc(sizeof(AlphaType)\*6);

inputmake(A);

printf("freq\t\n");

for(i=0; i<6; i++){

printf("%c %d\n",A[i].alpha,A[i].freq);

}

printf("Huffman code\t\n");

HuffmanTree(A,6);

printf("input String : ");

scanf("%s",string);

printf("\n");

for(i=0; i<8; i++)

HuffmanTreeC(A,6,string[i]);

printf("\n");

}