**第五次实验报告**

14307110190 周天树

**实验一：**赫夫曼树

**实验代码：**

#include<iostream>

#include<cstring>

using namespace std;

struct HuffmanNode{ //Huffman树的基本定义

int value;

int parent;

int left,right;

int depth;

}Huffman[1000];

void MovingTree(int index, int deep){ //计算各叶节点的深度

Huffman[index].depth = deep;

if(Huffman[index].left > -1){

MovingTree(Huffman[index].left, deep + 1);

}

if(Huffman[index].right > -1){

MovingTree(Huffman[index].right, deep + 1);

}

return;

}

int main(){

int n,i,weight,ans = 0;

cin >> n;

memset(Huffman,0,sizeof(HuffmanNode));

for(int j = 0; j < n; j++){

cin >> weight;

Huffman[j].value = weight;

}

for(int j = 0; j < 2 \* n + 1; j++){

Huffman[j].left = Huffman[j].right = -1;

}

/\* 以下循环为计算各个父节点的信息 \*/

for(i = n; i < 2 \* n - 1; i++){

int min1 = 100000, min2 = 100000;

int minIndex1 = -1, minIndex2 = -1;

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

if(Huffman[j].parent){

continue;

}

if(Huffman[j].value < min1){

min2 = min1;

minIndex2 = minIndex1;

min1 = Huffman[j].value;

minIndex1 = j;

}

else if(Huffman[j].value < min2){

min2 = Huffman[j].value;

minIndex2 = j;

}

}

Huffman[minIndex1].parent = i;

Huffman[minIndex2].parent = i;

Huffman[i].value = Huffman[minIndex1].value + Huffman[minIndex2].value;

Huffman[i].left = minIndex1;

Huffman[i].right = minIndex2;

}

/\* 循环结束 \*/

MovingTree(i - 1, 0);

for(int j = 0; j < n; j++){ //计算总权值

ans += Huffman[j].value \* Huffman[j].depth;

}

cout << ans << endl;

return 0;

}

**实验思路：**

本题的主要思路是先设置Huffman树的结点结构，并将其定义为一个数组以承载Huffman树的所有结点。再通过不断找寻数组中已有的结点内的两个没有处理过的最小权值结点，不断将其连接建树，从而将Huffman树还原。

还原Huffman树之后，利用递归，从树的最顶端根结点不断向下递归以求出各个叶结点的深度。再通过求“深度\*权值”的总和，来算出此Huffman树中所有叶结点的总权值。

-------------------------------------------------------------------------------------------------------

**实验二：**广义表操作

**实验代码：**

#include<iostream>

using namespace std;

/\* 本题中没有使用数据结构，直接采用枚举法枚举各种可能情况（数括号，详细分析见实验报告），暴力求解 \*/

char str[1000];

int main(){

int n;

int cur = 0; //前置标记点，标记指定字串的起始点

int end = 0; //后缀标记点，标记指定字串的终止点

int k = cur;

char opt;

cin >> str;

while(str[end++]); //后缀标记点初始化

cin >> n;

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

cin >> opt;

if(opt == 'T'){ //Tail情况下将前置标记点向前，直到掠过Head元素结束

int count = 0;

while(str[k]){

if(str[k] == '('){

count++;

}

else if(str[k] == ')'){

count--;

}

else if(str[k] == ','){

if(count == 1){

break;

}

else if(count == 0){

break;

}

}

k++;

}

cur = k;

cur++;

k++;

}

else if(opt == 'H'){ //Head情况下将后缀标记点前移，至Head元素之末尾结束

int count = 0;

while(str[k]){

if(str[k] == '('){

count++;

}

else if(str[k] == ')'){

count--;

}

else if(str[k] == ','){

if(count == 0){

break;

}

}

k++;

}

end = k;

k = cur;

}

}

for(int i = cur; i < end; i++){

if(str[i] != '(' && str[i] != ')'){ //顺序输出前置——后缀范围内的所有非括号字符，即为所求元素

cout << str[i];

}

}

cout << endl;

return 0;

}

**实验思路：**

开始时本人尝试使用重建广义表的方法解此题，后来发现这样实在太麻烦，因为本人对广义表这一数据结构不甚清楚，且课本上对于此数据结构的描述也是模糊不清。因此本人权衡之后使用暴力枚举数括号的方法求解。

本人的思路是这样的：置两标记点：前置标记点与后缀标记点，分别代表符合条件的范围的起始位置与终止位置。初始时前置标记点下标为0，后缀标记点下标为广义表字符串总长度(N) – 1.

完成初始化后，开始读取操作序列:

当读到操作为T(Tail)时，将前置标记点前置至掠过Head元素为止。判断掠过过程结束的标志为：越过的左括号数目-越过的右括号数目 = 1（例如**(apple,(pear,(banana),orange))**中的第一次T操作）或0（例如**(((apple)),((pear)),(banana),orange)**的第二次T操作）。

当读到操作为H(Head)时，将后缀标记点前移至移到Head元素尾端位置。判断前移过程结束的标志位：越过的左括号数目-越过的右括号数目 = 0.

执行所有T与H操作终了时，顺序输出前置标记点与后缀标记点间的所有符合要求的字符，即为所求元素！

**另附上本人第一次设计失败的代码：**

#include<iostream>

#include<stack>

#include<string>

using namespace std;

enum ElemTag{A,L};

struct GNode{

ElemTag Tag;

union{

char data;

struct{

GNode \*head;

GNode \*tail;

}ptr;

};

};

string Cut(string &S){

int n,i,k;

string ch,h;

n = S.length();

for(i = 0,k = -1; i < n && (ch != "," || k != 0); i++){

ch = S.substr(i,1);

if(ch == "("){

k++;

}

else if(ch == ")"){

k--;

}

}

if(i < n){

h = S.substr(1,i - 2);

S = "(" + S.substr(i,n - i);

}

else{

h = S.substr(1,n - 2);

S = "";

}

return h;

}

void Create(GNode \*&G, string S){

string Sub;

if(S == "()"){

G = NULL;

}

else{

G = new GNode;

if(S.length() == 1){

G->Tag = A;

G->data = S[0];

}

else{

G->Tag = L;

Sub = Cut(S);

Create(G->ptr.head,Sub);

if(S.empty()){

G->ptr.tail = NULL;

}

else{

Create(G->ptr.tail,S);

}

}

}

}

void Traverse(GNode \*G){

if(G == NULL){

cout << "()";

}

else{

if(G->Tag == A){

cout << G->data;

}

else{

GNode \*p = NULL;

cout << "(";

p = G;

while(p){

Traverse(p->ptr.head);

p = p->ptr.tail;

if(p){

cout << ",";

}

cout << ")";

}

}

}

}

GNode\* GetHead(GNode \*G){

return G->ptr.head;

}

GNode\* GetTail(GNode \*G){

return G->ptr.tail;

}

int main(){

int n;

char temp;

GNode \*G = NULL;

string S;

cin >> S;

Create(G,S);

GNode \*t = G;

cin >> n;

/\*for(int i = 0; i < n; i++){

cin >> temp;

if(temp == 'T'){

t = GetTail(t);

}

if(temp == 'H'){

t = GetHead(t);

}

}

while(t->ptr.head->Tag == A){

cout << t->data;

t = t->ptr.head;

}\*/

Traverse(G);

cout <<endl;

return 0;

}