**1.栈的应用**

#include<iostream>

#include<stack>

#include <ctime>

#include<stdlib.h>

using namespace std;

void printfmaze(int maze[10][10])

{

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

cout << maze[i][j] << " ";

cout.setf(ios\_base::unitbuf);

}

cout << endl;

}

}//打印输出

void Delay(int time)//time\*1000为秒数

{

clock\_t now = clock();

while (clock() - now < time);

}

int main()

{

int maze[10][10] = {

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2},

{2, 0, 0, 0, 0, 0, 0, 0, 0, 2},

{2, 0, 0, 0, 0, 0, 0, 0, 0, 2},

{2, 0, 0, 0, 0, 0, 0, 0, 0, 2},

{2, 0, 0, 0, 0, 0, 0, 0, 0, 2},

{2, 0, 0, 0, 0, 0, 0, 0, 0, 2},

{2, 0, 0, 0, 0, 0, 0, 0, 0, 2},

{2, 0, 0, 0, 0, 0, 0, 0, 0, 2},

{2, 0, 0, 0, 0, 0, 0, 0, 0, 2},

{2, 2, 2, 2, 2, 2, 2, 2, 2, 2}

};

int x, y, z;

x = 1; y = 1; z = 0;

stack<int>stk;

stk.push(x);

stk.push(y);//读取初始位置

while (!stk.empty())

{

y = stk.top();

stk.pop();

x = stk.top();

stk.push(y);//从栈中读取坐标

maze[x][y] = 1;

if (maze[x - 1][y] == 0){

stk.push(x-1);

stk.push(y);

}

else if (maze[x + 1][y] == 0) {

stk.push(x + 1);

stk.push(y);

}

else if (maze[x][y - 1] == 0) {

stk.push(x);

stk.push(y - 1);

}

else if (maze[x][y + 1] == 0) {

stk.push(x);

stk.push(y + 1);

}//移动

else {

stk.pop();

stk.pop();

}//如有阻碍就退回一步

printfmaze(maze);

Delay(500);

cout << endl;

for (int i = 0; i < 10; i++) {

for (int j = 0; j < 10; j++) {

z = z + maze[i][j];

}

}

if (z==136)

{

exit(100);

}

else

{

z = 0;

}

}//判断是否填满

return 0;

}

2. 二叉树的应用－哈夫曼编码的实现

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<cstring>

#include<cstdio>

#include<fstream>

#include<string>

using namespace std;

typedef struct {

int weight;

int parent, lchild, rchild;

}HTNode, \* HuffmanTree;

typedef char\*\* HuffmanCode;

int a[24 + 1], b[24 + 1];

char c[24 + 1];

void Select(HuffmanTree& HT, int n, int& s1, int& s2) {

for (int i = 1; i <= n; i++)

if (HT[i].parent == 0 && s1 == 0) { s1 = i; break; }

for (int i = 1; i <= n; i++)

if (HT[i].parent == 0 && HT[i].weight < HT[s1].weight) s1 = i;

for (int i = 1; i <= n; i++)

if (HT[i].parent == 0 && s2 == 0 && i != s1) { s2 = i; break; }

for (int i = 1; i <= n; i++)

if (HT[i].parent == 0 && HT[i].weight < HT[s2].weight && i != s1) s2 = i;

}

void CreateHuffmanTree(HuffmanTree& HT, int n) {

if (n <= 1) return;

int m = 2 \* n - 1;//构造一棵哈夫曼树所需要的所有节点数

HT = new HTNode[m + 1];//m+1指从第一个结点开始，第0个结点不用

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

HT[i].parent = 0;

HT[i].lchild = 0;

HT[i].rchild = 0;

}

for (int i = 1; i <= n; i++) HT[i].weight = b[i];//把字母出现的次数赋值给哈夫曼树的weight

for (int i = n + 1; i <= m; i++) {

int s1 = 0, s2 = 0;

Select(HT, i - 1, s1, s2);

HT[s1].parent = i;

HT[s2].parent = i;

HT[i].lchild = s1;

HT[i].rchild = s2;

HT[i].weight = HT[s1].weight + HT[s2].weight;

}

}

void CreateHuffmanCode(HuffmanTree& HT, HuffmanCode& HC, int n) {

HC = new char\* [n + 1];

char\* cd = new char[n];

cd[n - 1] = '\0';

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

int start = n - 1, c = i, f = HT[i].parent;

while (f) {

--start;

if (HT[f].lchild == c) cd[start] = '0';

else cd[start] = '1';

c = f;

f = HT[f].parent;

}

HC[i] = new char[n - start];

strcpy(HC[i], &cd[start]);

}

//delete cd;

}

int main() {

freopen("SourceFile.txt", "r", stdin);//打开输入文件

memset(a, 0, sizeof(a));

string str;

getline(cin, str);//接收文件的字符串并保存到str中

for (int i = 0; i < str.length(); i++)//统计每个字母出现的频率即次数

a[str[i] - 64]++;

int n = 0;

for (int i = 1, j = 1; i <= 24; i++)//统计有多少个不同的字母以及把没出现的字母去除掉

if (a[i] != 0) {

n++;

b[j] = a[i];

c[j++] = i + 64;

}

HuffmanTree HT;

HuffmanCode HC;

CreateHuffmanTree(HT, n);

CreateHuffmanCode(HT, HC, n);

freopen("Code.txt", "w", stdout);

for (int i = 1; i <= n; i++)

cout << c[i] << ':' << HC[i] << endl;

freopen("ResultFile.txt", "w", stdout);

for (int i = 0; i < str.length(); i++) {

for (int j = 1; j <= n; j++) {

if (str[i] == c[j]) cout << HC[j];

}

}

return 0;

}

3. 基数排序

#include <iostream>

using namespace std;

int maxdigits(int arr[],int n)

{

int a, b, i;

a = 1; b = 10; i = 0;

while (i<n)

{

if (arr[i]>b)

{

a++;

b = b \* 10;

i = 0;

}

else

{

i++;

}

}

return a;

}

void radixsort(int data[], int n)

{

int d = maxdigits(data, n);

int\* tmp = new int[n];//申请动态内存

int k = 0;

int count[10];

int radix = 1;

for (int i = 1; i <= d; i++) //进行d次排序

{

for (int j = 0; j < 10; j++)

count[j] = 0; //清空计数器

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

{

k = (data[j] / radix) % 10;

count[k]++; //统计对应桶中含有的数字个数

}

for (int j = 1; j < 10; j++)

count[j] = count[j - 1] + count[j]; //将tmp中的位置依次分配给每个桶，即确定每个桶中最后一个数在tmp中的位置

for (int j = n - 1; j >= 0; j--)

{

k = (data[j] / radix) % 10;//确定基数

tmp[count[k] - 1] = data[j];//放入数字

count[k]--;//向前移一位

}

for (int j = 0; j < n; j++) //将临时数组的内容复制到data中

data[j] = tmp[j];

radix = radix \* 10;

}

delete tmp;

}

int main()

{

int i;

cout << "请输入数组长度:";

cin >> i;

cout << endl;

int\* array = new int[i];

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

{

cout << "请输入第" << m + 1 << "项数字:";

cin >> array[m];

cout << endl;

}//手动创建数组

maxdigits(array, i);

radixsort(array, i);

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

{

cout << array[n] << " ";

}

return 0;

}

5. 大小堆排序

#include<cstdio>

#include<iostream>

#include<cstring>

#include<algorithm>

using namespace std;

void compare(int arr[], int len, int index)

{

int left = 2 \* index + 1;//代表index节点的左孩子

int right = 2 \* index + 2;//代表index节点的右孩子

int maxIdx = index;//父节点

if (left<len && arr[left] > arr[maxIdx])

{

maxIdx = left;

}

if (right<len && arr[right] > arr[maxIdx])

{

maxIdx = right; // maxIdx是3个数中最大数的下标

}

if (maxIdx != index)// 如果maxIdx的值有更新

{

swap(arr[maxIdx], arr[index]);

compare(arr, len, maxIdx);// 父节点改变导致其堆中大小关系需要重新调整，故递归调整其不满足堆性质的部分

}

}

void locating(int arr[], int size)

{

for (int i = size / 2 - 1; i >= 0; i--) // 对每一个非叶结点进行堆调整(从最后一个非叶结点开始)

{

compare(arr, size, i);

}

for (int i = size - 1; i >= 1; i--)

{

swap(arr[0], arr[i]);// 将当前最大的放置到数组末尾

compare(arr, i, 0);// 将未完成排序的部分继续进行堆排序，且无需调整最后一个数字

}

}

int main()

{

int i;

cout << "请输入数组长度:";

cin >> i;

cout << endl;

int\* array = new int[i];

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

{

cout << "请输入第" << m + 1 << "项数字:";

cin >> array[m];

cout << endl;

}//手动创建数组

locating(array, i);

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

{

cout << array[n] << " ";

}

return 0;

}

6.多项式相加

#include <iostream>

using namespace std;

typedef struct node /\*定义结构体\*/

{

int factor; //系数

int indice; //幂

struct node\* next;

}node, \* LinkList;

void CreateLink(LinkList& L, int n)

{

LinkList p, s;

int i;

L = (LinkList)malloc(sizeof(node)); //头结点

L->next = NULL;

p = L;

for (i = 0; i < n; i++) //尾插法建链表

{

s = (LinkList)malloc(sizeof(node));

cout << "请输入第" << i + 1 << "项的系数和幂：" ;

cin >> s->factor >> s->indice;

s->next = NULL;

p->next = s;

p = s;

}

}

void AddList(LinkList List1, LinkList List2, LinkList& L)

{

LinkList p1, p2, list, s;

L = (LinkList)malloc(sizeof(node)); //建合并后新链表的头结点

L->next = NULL;

list = L;

p1 = List1->next;

p2 = List2->next;

while (p1 && p2) //p1,p2两个链表不空时循环

{

if (p1->indice < p2->indice) //p1的幂小于p2的幂，将p2结点加入到list链表中

{

s = p2->next; //找到要断开结点的后继结点

list->next = p2; //链入

list = p2;

list->next = NULL; //断开

p2 = s;

}

else if (p1->indice == p2->indice) //p1的幂等于p2的幂

{

p1->factor = p1->factor + p2->factor;

if (p1->factor != 0)//系数和不为0

{

s = p1->next;

list->next = p1;

list = p1;

list->next = NULL;

p1 = s;

p2 = p2->next;

}

else //系数和为0，继续向后找

{

p1 = p1->next;

p2 = p2->next;

}

}

else //p1的幂大于p2的幂，将p1结点加入到list链表中

{

s = p1->next;

list->next = p1;

list = p1;

list->next = NULL;

p1 = s;

}

}

if (p1 != NULL) list->next = p1;

if (p2 != NULL) list->next = p2;

}

void VisitList(LinkList L)

{

LinkList p;

p = L->next;

if (p == NULL)

{

cout << "0" << endl;

}

while (p)

{

cout << p->factor << "X^" << p->indice;

p = p->next;

if (p != NULL)

cout << "+";

}

cout << endl;

}

int main()

{

LinkList List1, List2, L;

int num;

cout << "输入第一个多项式节点个数:";

cin >> num;

CreateLink(List1, num);

cout << "第一个多项式为：";

VisitList(List1);

cout << "输入第二个多项式节点个数:";

cin >> num;

CreateLink(List2, num);

cout << "第二个多项式为：";

VisitList(List2);

cout << "多项式合并结果为：";

AddList(List1, List2,L);

VisitList(L);

system("pause");

return 0;

}