第五章：

1. 在以行序为主序的存储结构中，给出三维数组A2\*3\*4的地址计算公式（下标从0开始计数）。

LOC(aijk) = LOC(a111) + (i – 0)\* 3 \* 4 + (j – 0) \* 4 + (k – 0)

2. 数组A中，每个元素A的长度均为32个二进位，行下标从-1到9，列下标从1到11，从首地址s开始连续存放主存储器中，主存储器字长为16位。求：

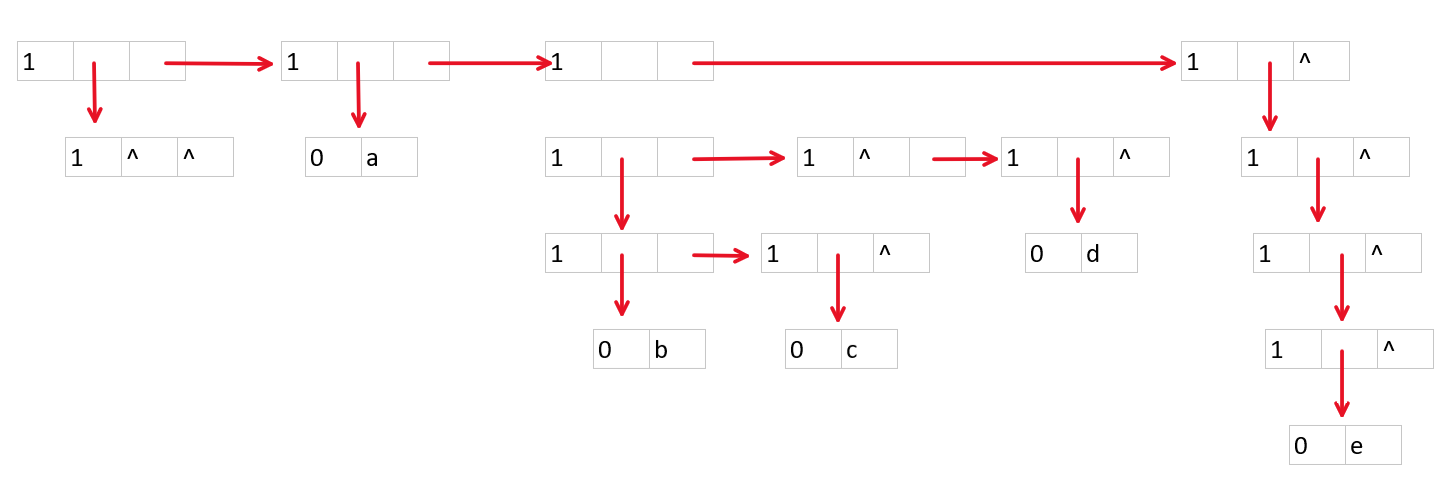
(1)存放该数组所需多少单元？ 242

(2)存放数组第4列所有元素至少需多少单元？ 22

(3)数组按行存放时，元素A[7，4]的起始地址是多少？ s + 182

(4)数组按列存放时，元素A[4，7]的起始地址是多少？ s + 142

3. 画出下列广义表的链接存储结构，并求其深度： ((((),a,((b,c),(),d),(((e))))



深度为4

4. 设有广义表K1(K2(K5(a，K3(c，d，e))，K6(b，k))，K3，K4(K3，f))，要求：

(1)指出K1的各个元素及元素的构成。

K2 : K5, K6

K3 : c, d, e

K4 : K3, f

(2)计算表K1，K2，K3，K4，Ks，K6的长度和深度。

K1 : 3, 4;

K2 : 2, 3;

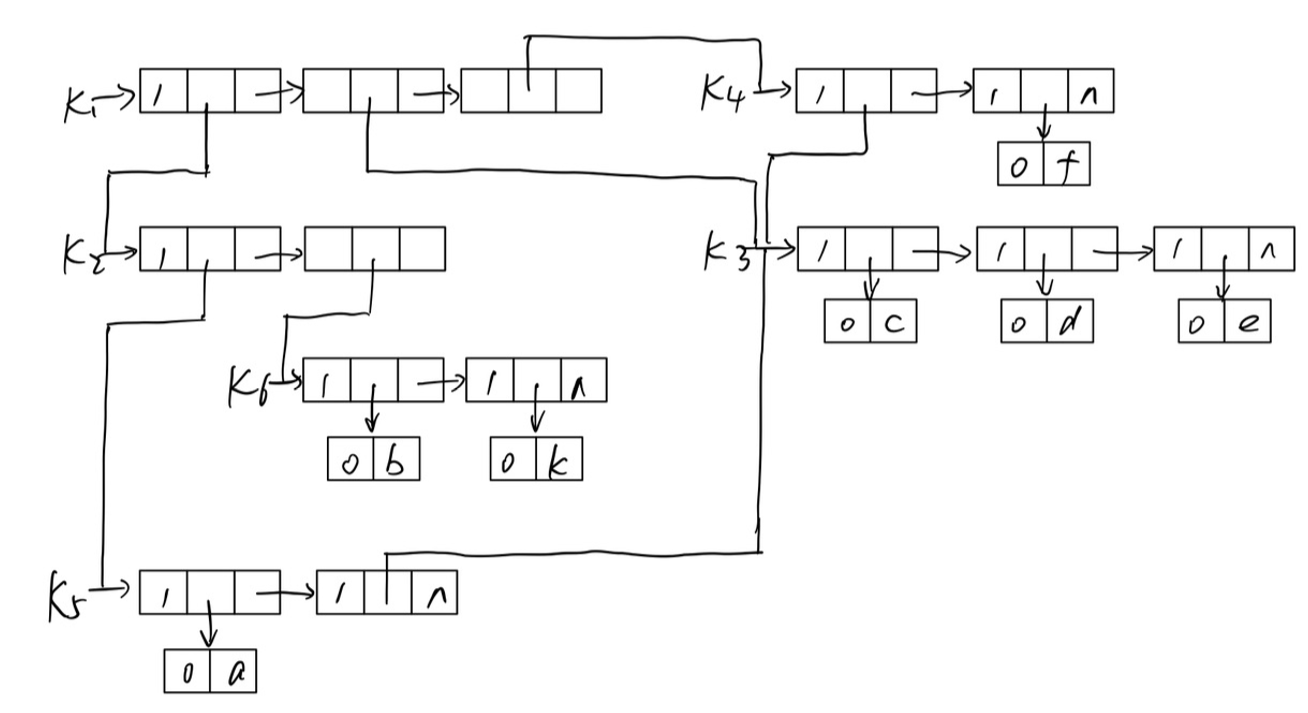
K3 : 3, 1;

K4 : 2, 2;

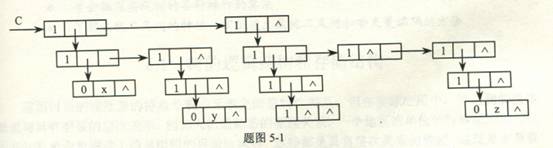
K5 : 2, 2;

K6 : 2, 1;

(3)画出K1的链表存储结构。



5. 已知下图为广义表的链接存储结构，写出该图表示的广义表。



((x,(y)),((()),(),(z)))

6. 数组A[1..8，-2..6，0..6]以行为主序存储，设第一个元素的首地址是78，每个元素的长度为4，试求元素A[4,2,3]的存储首地址。

LOCA = 78 + 4 \* [(4 – 1) \* 9 \* 7 + (2 + 2) \* 7 + (3 – 0) = 958

7. 特殊矩阵和稀疏矩阵哪一种压缩存储后失去随机存取的功能？为什么？

稀疏矩阵

特殊矩阵的元素存在一定规律，因此可讲非零元素存于一维数组中，使得其下表与数组中下表存在规律，从而实现随机存取。

稀疏矩阵的元素不存在规律，且存在大量零元素。使用三元组法存储时，须扫描其下标，不同元素所需时间不同；使用十字链表法，则为链式结构，自然无法实现随机存取。

8. 数组，广义表与线性表之间有什么样的关系？

数组是有序的元素序列，为计算机语言自带的数据类型。

广义表与线性表则属于数据结构，为一有限序列。线性表中的元素被限定为单个元素，可以以顺序存储结构和链式存储结构存储；而广义表中的元素既可以是单个元素，也可以为一子广义表，因此只能以链式存储结构存储。

9. 设有三对角矩阵(aij)n\*n，将其三条对角线上的元素逐行地存于数组B(1:3n-2)中，使得B[k]=aij，求：

（1）用i,j表示k的下标变换公式；

i, j, k从1开始

k = 2 \* i + j – 2;

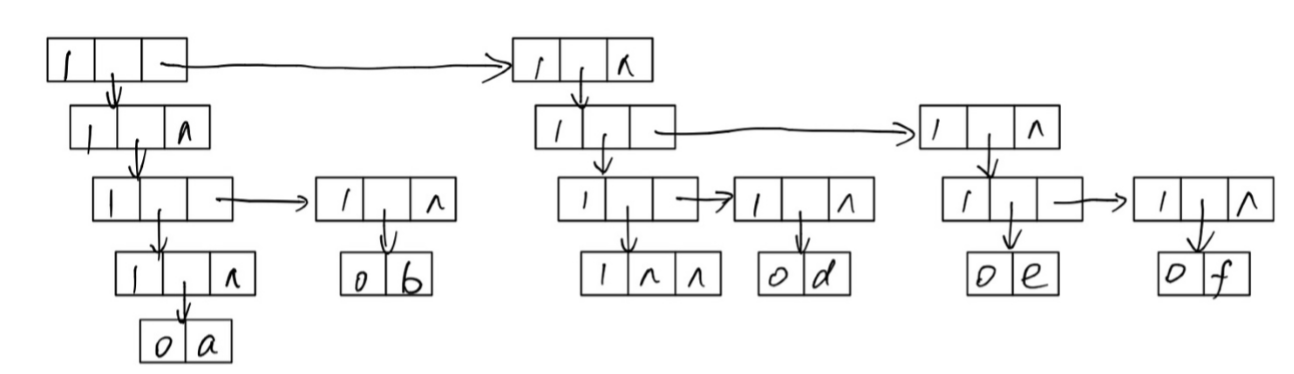
（2）用k表示i,j的下标变化公式。

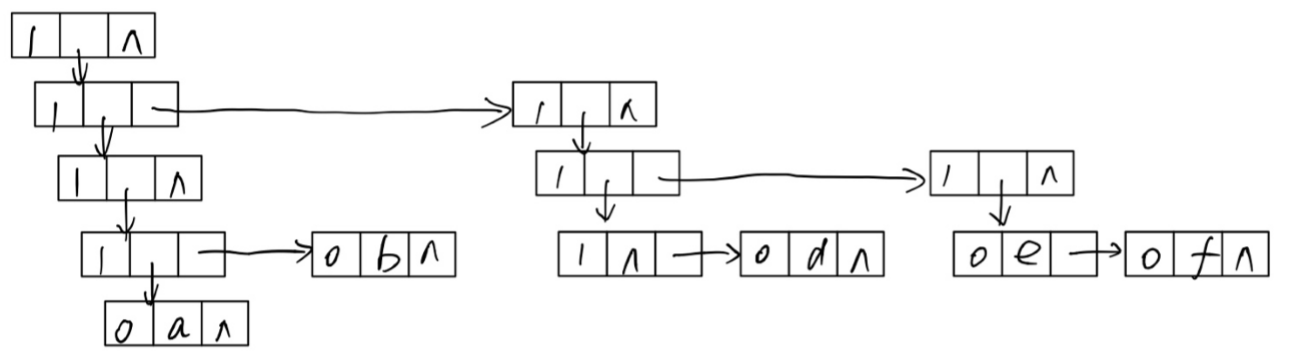
i = k / 3 + 1;

j = k / 3 + k % 3;

10．画出下面广义表的两种存储结构图示：

((((a), b)), ((( ), d), (e, f)))





11．求下列广义表运算的结果：

（1）HEAD[((a,b),(c,d))]; ((a,b))

（2）TAIL[((a,b),(c,d))]; ((c, d))

（3）TAIL[HEAD[((a,b),(c,d))]]; (b)

（4）HEAD[TAIL[HEAD[((a,b),(c,d))]]]; b

（5）TAIL[HEAD[TAIL[((a,b),(c,d))]]]; d

1. 利用广义表的Head和Ｔail运算，把原子d分别从下列广义表中分离出来，L1＝(((((a),b),d),e))；L2＝（a,(b,((d)),e)）。

HEAD[TAIL[HEAD[HEAD(L1)]]

HEAD[HEAD[HEAD[TAIL[TAIL(L2)]]]]

13. 对于二维整型数组A[m,n]，分别编写相应函数实现如下功能：

    (1)求数组A4边元素之和。

int sumEdge(int\*\* A, int m, int n)

{

int result = 0;

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

result += A[0][i];

}

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

result += A[i][0];

result += A[i][n - 1];

}

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

result += A[m - 1][i];

}

return result;

}

    (2)当m=n时分别求两条对角线上的元素之和，否则显示m≠n的信息。

void ifEqual(int\*\* A, int m, int n)

{

if (m != n) {

cout << "m != n" << endl;

return ;

}

int diag1 = 0;

int diag2 = 0;

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

diag1 += A[i][i];

}

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

diag2 += A[i][n - 1 - i];

}

cout << "diag1 = " << diag1 << endl

<< "diag2 = " << diag2 << endl;

return;

14. 编写一个函数将两个广义表合并成一个广义表。合并是指元素的合并，如两个广义表((a，b)，(c))与(a，(e，f))合并后的结果是((a，b)，(c)，a，(e，f））。

int equal(GListNode\* ha, GListNode\* hb);

void GListInsertTail(GListNode\* h, GListNode\* s);

GListNode\* GListMerge(GListNode\* ha, GListNode\* hb) {

GListNode p, q, r;

if (!ha) {

ha = hb;

return ha;

}

else {

p = hb->val.sublist;

while (p) {

q = ha->val.sublist;

while (q) {

if (equal(p, q))

break;

q = q->link;

}

if (!q) {

r = p->link;

GListInsertTail(ha, p);

p = r;

}

else

p = p->link;

}

}

return ha;

}

int equal(GListNode\* ha, GlistNode\* hb) {

GListNode\* p，\* q;

if (!ha) {

if (!hb)

return 1;

else

return O;

}

if (ha->tag != hb->tag)

return O;

if (!ha->tag) {

if (ha->val.data == hb->val.data)

return l;

else

return O;

}

else {

p = ha->val.sublist;

q = hb->val.sublist;

while (p && q && equal(p, q)) {

p = p->link;

q = q->link;

}

if (!p && !q)

return l;

else

return O;

}

}

void GlistInsertTail(GListNode\* h, GListNode\* s) {

GListNode p, r;

p = h->val.sublist;

r = h;

while (p) {

r = p;

p = p->link;

}

r->link = s;

s->link = NULL;

}

15. 设二维数组a[1..m, 1..n] 含有m\*n 个整数。

(1) 写出算法：判断a中所有元素是否互不相同?输出相关信息(yes/no)；

void isSame(int\*\* A, int m, int n)

{

int amount = m \* n;

int\* one = new int[amount];

int pos = 0;

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

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

one[pos++] = A[i][j];

}

}

sort(one, one + amount);

bool flag = 1;

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

if (one[i] == one[i + 1]) {

flag = 0;

break;

}

}

if (flag == 0)

cout << "no" << endl;

else

cout << "yes" << endl;

delete[] one;

return;

}

(2) 试分析算法的时间复杂度。

O(m \* n)

16．稀疏矩阵用三元组的表示形式，试写一算法实现两个稀疏矩阵相加，结果仍用三元组表示。

SPMatrix add(const SPMatrix& a, const SPMatrix& b)

{

SPMatrix result = a + b;

}

SPMatrix SPMatrix::operator+(const SPMatrix& b){

//矩阵加法为每行依次相加

//此段以this表示data所属的矩阵

SPMatrix result(mu, nu);

int posA = 1;

int posB = 1;

int posR = 1;

int valueR = 0; //result的value，若不为0则存入

while (posA <= this->tu && posB <= b.tu) {

//位于同一行

if (this->data[posA].row == b.data[posB].row) {

if (this->data[posA].col < b.data[posB].col) {

result.data[posR].row = this->data[posA].row;

result.data[posR].col = this->data[posA].col;

result.data[posR].value = this->data[posA].value;

posA++;

posR++;

}//if

else if (this->data[posA].col > b.data[posB].col) {

result.data[posR].row = b.data[posB].row;

result.data[posR].col = b.data[posB].col;

result.data[posR].value = b.data[posB].value;

posB++;

posR++;

}//else if

else {

valueR = this->data[posA].value + b.data[posB].value;

//非零则存入

if (valueR != 0) {

result.data[posR].row = this->data[posA].row;

result.data[posR].col = this->data[posA].col;

result.data[posR].value = valueR;

posR++;

}//if

posA++;

posB++;

}//else

}//if 同一行

else if (this->data[posA].row < b.data[posB].row) {

result.data[posR].row = this->data[posA].row;

result.data[posR].col = this->data[posA].col;

result.data[posR].value = this->data[posA].value;

posA++;

posR++;

}//else if A在前一行

else {

result.data[posR].row = b.data[posB].row;

result.data[posR].col = b.data[posB].col;

result.data[posR].value = b.data[posB].value;

posB++;

posR++;

}//else B在前一行

result.tu = posR;

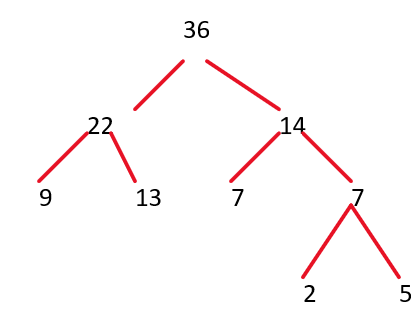
}//while

return result;

}//operator+

第六章：

1.以数据集合{2,5,7,9,13}为权值构造一棵赫夫曼树，并计算其带权路径长度



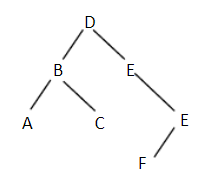
79

2.设二叉树根节点所在层次为1，树的深度d为距离根节点最远的叶节点所在的层次，分别求出深度为d的完全二叉树和满二叉树的不同二叉树棵数。

完全二叉树 ： 2 ^ (d – 1);

满二叉树 ： 1

3.给定如图二叉树，写出其三序遍历结果。



前序遍历 ： DBACEGF

中序遍历 ： ABCDEFG

后序遍历 ： ACBFGED

4.高度为L的满K叉树有以下性质：第L层上的结点都是叶子结点，其余各层上的每个结点都有K棵非空子树，如果从上到下，自左至右，对K叉树中全部结点进行编号（根结点为1），求：

（1）各层结点的数目。

第i层 ：K ^ (i – 1)

（2）编号为n的结点的双亲结点（若存在）。

(n + K – 2) / K

（3）第i个孩子结点（若存在）。

提示：K叉树中编号为i的结点，从右向左数第二个孩子结点的编号为K×i。

K \* (n – 1) + i + 1

5.假设二叉树采用二叉链存储结构，设计一个算法，计算一棵给定二叉树的所有结点数。

int countNode(BinTreeNode\* T)

{

if (!T)

return 0;

else

return 1 + countNode(T->leftChild) + countNode(T->rightChild);

}

6.假设二叉树采用二叉链存储结构，设计一个算法，计算一棵给定二叉树的所有叶子结点数。

int countLeaf(BinTreeNode\* T)

{

if (!T)

return 0;

else {

if (!T->leftChild && !T->rightChild)

return 1;

else

return countLeaf(T->leftChild) + countLeaf(T->rightChild);

}

}

7.写出求二叉树深度的算法

int height(BinTreeNode\* T)

{

if (!T)

return 0;

else {

int l = height(T->leftChild);

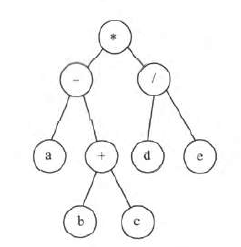
int r = height(T->rightChild);

return l > r ? l + 1 : r + 1;

}

}

8.表达式（a-（b+c））\*（d/e）存储在如图所示的一棵以二叉链表为存储结构的二叉树中（二叉树结点的data域为字符型），编写程序求出该表达式的值（表达式中的操作数都是一位整数）



void get\_m(char\* t, Node\* T, int& k) const

{

if (T) {

get\_m(t, T->leftChild, k);

t[k] = t->value;

k++;

get\_m(t, T->rightChild, k);

}

}

char\* traversal\_m() const {

char\* p = new int[len];

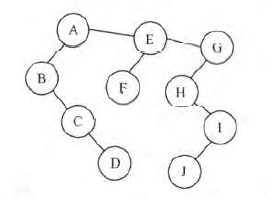
int i = 0;

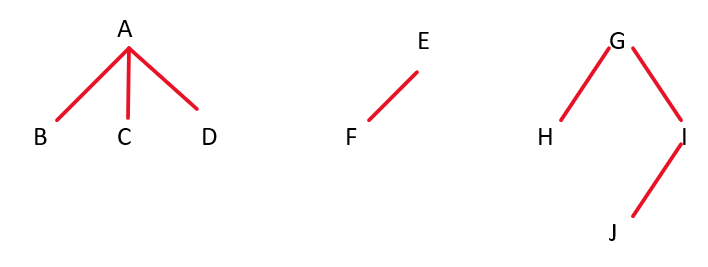
get\_m(p, Bin, i);

return p;

}

9.将以下二叉树转化为森林。





10.已知一棵有2011个结点的树，其叶结点有116个，求该树对应的二叉树中无右孩子的结点个数。

N0 + N1 + N2 = 2011;

N0 = 116;

N0 = N2 + 1;

N1 = 1780;

所以所求为1780 + 116 = 1896