存储结构分为如下四种：

（1）顺序存储方法

（2）链式存储方法

（3）索引存储方法

（4）散列存储方法

算法的五个重要的特性 ：

（1）有穷性

（2）确定性

（3）可行性

（4）有输入

（5）有输出

对于空间复杂度为O(1)的算法称为原地工作或就地工作算法。

顺序表基本运算的实现

typedef struct

{ ElemType elem[MaxSize]; /\*存放顺序表元素\*/

int length; /\*存放顺序表的长度\*/

} SqList;

插入数据元素算法：元素移动的次数不仅与表长n有关 ；插入一个元素时所需移动元素的平均次数 n/2。平均时间复杂度为O(n)。

删除数据元素算法:元素移动的次数也与表长n有关 。删除一个元素时所需移动元素的平均次数为(n-1)/2。删除算法的平均时间复杂度为O(n)。

定义单链表结点类型:

typedef struct LNode

{ ElemType data;

struct LNode \*next; /\*指向后继结点\*/

} LinkList;

定义双链表结点类型:

typedef struct DNode

{ ElemType data;

struct DNode \*prior; /\*指向前驱结点\*/

struct DNode \*next; /\*指向后继结点\*/

} DLinkList;

采用头插法建表的算法如下:

void CreateListF(LinkList \*&L,ElemType a[],int n)

{ LinkList \*s;int i;

L=(LinkList \*)malloc(sizeof(LinkList)); /\*创建头结点\*/

L->next=NULL;

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

{ s=(LinkList \*)malloc(sizeof(LinkList));

/\*创建新结点\*/

s->data=a[i]; s->next=L->next;

/\*将\*s插在原开始结点之前,头结点之后\*/

L->next=s;

}

}

尾插法

void CreateListR(LinkList \*&L,ElemType a[],int n)

{ LinkList \*s,\*r;int i;

L=(LinkList \*)malloc(sizeof(LinkList));

/\*创建头结点\*/

r=L; /\*r始终指向终端结点,开始时指向头结点\*/

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

{ s=(LinkList \*)malloc(sizeof(LinkList));

/\*创建新结点\*/

s->data=a[i];r->next=s; /\*将\*s插入\*r之后\*/

r=s;

}

r->next=NULL; /\*终端结点next域置为NULL\*/

}

设一个整数线性表采用带头节点的hc单链表存放，设计一个算法在不破坏原有单链表的前提下，创建两个单链表ha和hb，前者由hc中值为奇数的节点构成，后者由hc中值为偶数的节点构成。

void split(LinkList \*hc, LinkList \*&ha, LinkList \*&hb)

{ LinkList \*p=hc->next,\*ra,\*rb,\*s;

ha=(LinkList \*)malloc(sizeof(LinkList));

ra=ha;

hb=(LinkList \*)malloc(sizeof(LinkList));

rb=hb;

while (p!=NULL)

{ if (p->data%2==1) //奇数节点

{ s=(LinkList \*)malloc(sizeof(LinkList));

s->data=p->data; //复制产生节点\*s

ra->next=s; //将\*s节点链接到ha尾部

ra=s;

}

else //偶数节点

{ s=(LinkList \*)malloc(sizeof(LinkList));

s->data=p->data; //复制产生节点\*s

rb->next=s; //将\*s节点链接到hb尾部

rb=s;

}

p=p->next;

}

ra->next=rb->next=NULL; //两链表尾节点next均置为NULL

}

顺序栈

typedef struct

{ ElemType data[MaxSize];

int top; /\*栈顶指针\*/

} SqStack;

顺序栈的4要素

栈空条件:s.top==-1

栈满条件:s.top==MaxSize-1

进栈:top++;s.data[s.top]=e;

出栈:e=s.data[s.top];s.top—;

链栈

typedef struct linknode

{ ElemType data; /\*数据域\*/

struct linknode \*next; /\*指针域\*/

} LiStack;

栈空条件:s->next==NULL

顺序队

typedef struct

{ ElemType data[MaxSize]；

int front,rear；/\*队首和队尾指针\*/

} SqQueue；

环形队列的4要素:

队空:q.front==q.rear

队满:(q.rear+1)%MaxSize==q.front

进队:q.rear=(q.rear+1)％MaxSize;q.data[q.rear]=e;

出队:q.front=(q.front+1)％MaxSize;e=q.data[q.front];

链队

struct qnode /\*数据结点\*/

{ ElemType data;

struct qnode \*next;

} QNode;

typedef struct /\*头结点\*/

{ QNode \*front;

QNode \*rear;

} LiQueue;

数组的存储结构

以数组A[c1..d1,c2..d2] 为例

以行序为主序 :

LOC(ai,j)=LOC(ac1,c2)+[(i-c1)\*(d2-c2+1)+(j-c2)]\*k

以列序为主序

LOC(ai,j)=LOC(ac1,c2)+[(j-c2)\*(d1-c1+1)+(i-c1)]\*k

对称矩阵

若一个n阶方阵A[n][n]中的元素满足ai,j=aj,i（0≤i，j≤n-1），则称其为n阶对称矩阵。

A[0..n-1][0..n-1] . B[0..n(n+1)/2]

i(i+1)/2 +j 当i≥j时

k=

j(j+1)/2 +i 当i＜j时

树的表示法（逻辑表示方法）

■ 树形表示法

■ 文氏图表示法

■ 凹入表示法

■ 括号表示法

二叉树：

性质1 非空二叉树上叶结点数等于双分支结点数加1。即n0=n2+1.

性质2 非空二叉树上第i层上至多有2i-1个结点（i≥1）。

性质3 高度为h的二叉树至多有2h-1个结点（h≥1） 。

性质4 完全二叉树的性质 。

性质5 具有n个（n＞0）结点的完全二叉树的高度为.log2(n+1).或.log2n.+1。

二叉链存储结构

typedef struct node

{ ElemType data; /\*数据元素\*/

struct node \*lchild; /\*指向左孩子\*/

struct node \*rchild; /\*指向右孩子\*/

} BTNode;

二叉树的遍历

■ 先序遍历

■ 中序遍历

■ 后序遍历

■ 层次遍历

假设二叉树采用二叉链存储结构存储，试设计一个算法，输出一棵给定二叉树的所有叶子结点。

解：输出一棵二叉树的所有叶子结点的递归模型f()如下：

f(b)：不做任何事件 若b=NULL

f(b)：输出\*b结点的data域 若\*b为叶子结点

f(b)：f(b->lchild);f(b->rchild) 其他情况

先序

void DispLeaf(BTNode \*b)

{

if (b!=NULL)

{

if (b->lchild==NULL && b->rchild==NULL)

printf("%c ",b->data);

DispLeaf(b->lchild);

DispLeaf(b->rchild);

}

}

哈夫曼树的定义

WPL最小，没有单分支结点即n1=0

深度优先搜索遍历

离初始点越远越优先访问。

void DFS(ALGraph \*G,int v)

{ ArcNode\*p;Visited[v]=1; /\*置已访问标记\*/

printf("%d ",v); /\*输出被访问顶点的编号\*/

p=G->adjlist[v].firstarc;

while (p!=NULL)

{ if (visited[p->adjvex]==0)

DFS(G,p->adjvex);

p=p->nextarc;

}

}

广度优先搜索遍历

离初始点越近越优先访问。

void BFS(ALGraph \*G,int v)

{ ArcNode \*p;

int queue[MAXV],front=0,rear=0;

int visited[MAXV]; int w,i;

for (i=0;i<G->n;i++) visited[i]=0;

printf("%2d",v);

visited[v]=1; /\*置已访问标记\*/

rear=(rear+1)%MAXV;

queue[rear]=v; /\*v进队\*/

while (front!=rear) /\*若队列不空时循环\*/

{ front=(front+1)%MAXV;

w=queue[front]; /\*出队并赋给w\*/

p=G->adjlist[w].firstarc;

while (p!=NULL)

{ if (visited[p->adjvex]==0)

{ printf("%2d",p->adjvex);

visited[p->adjvex]=1;

rear=(rear+1)%MAXV;

queue[rear]=p->adjvex;

}

p=p->nextarc;

}

}

}

最小生成树

■ 普里姆算法

给定起始点；

算法过程；

O(n2)

克鲁斯卡尔算法

不给定起始点；

算法过程；

O(elog2e)

最短路径

■ 狄克斯特拉算法过程

■ 弗洛伊德算法过程

（1）线性表的查找

模块4：其他

1. 查找

在顺序表上进行。方法有：

■ 顺序查找 （过程和算法）

■ 二分查找 （过程和算法）

■ 分块查找 (过程)

（2） 树表的查找

▲ 二叉排序树

■ 定义  与堆的区别

■ 查找（过程和算法）

■ 插入和删除（过程）

性质：二叉排序树的中序序列是一个有序序列

平衡二叉树

■ 定义

■ 查找（过程和算法）

■ 调整（过程）

哈希表查找

■ 哈希函数

主要有除留余数法。

■ 哈希冲突解决方法

主要有线性探查法、拉链法

2. 内排序

■ 插入排序

（1）直接插入排序

（2）希尔排序

■交换排序

（1）冒泡排序

（2）快速排序

■选择排序

（1）简单选择排序

（2）堆排序

■归并排序

■基数排序

在待排序的元素序列基本有序的前提下，效率最高的排序方法是 A。

A. 直接插入排序 B. 选择排序

C. 快速排序 D. 归并排序

快速排序在最坏情况下时间复杂度是O(n2)，比A 的性能差。

A. 堆排序 B. 冒泡排序

C. 直接选择排序 D. 直接插入排序

强调各种排序方法的比较 是否需要比较 排序方法的区别 时间复杂度 空间复杂度

外排序的过程。

磁盘排序：置换选择算法、败者树、最佳平衡归并树。