第二章 基本数据类型与数据结构

§2.1 线性表

一．什么是线性表？

线性表是由给定类型的0个或多个元素组成的序列：

\*线性表的一个重要性质是可以根据元素在表中的位置将它们线性地排序。

二．表上的操作：

1. Insert(x, p, L) : 在表L的位置p插入元素x。

2. Locate(x, L) : 确定元素x在表L中的位置。

3. Retrieve(p, L) : 返回表L中位置p的元素。

4. Delete(p, L) : 删除表L中位置p的元素。

5. Next(p, L)和Previous(p, L) : 返回表L中位置p的下一个(前一个)元素。

6. Makenull(L) : 把表L置为空表。

7. First(L) : 返回表L中第一个元素。

8. Printlist(L) : 顺序地输出表L中所有元素。

三．线性表的实现

1. 用数组实现

2. 用链表实现

(1) 简单链表

(2) 循环链表

(3) 双向链表

图2.1

四．链表的几个操作的算法

假设以下使用简单链表

PROCEDURE Insert (x : elementtype; p : position);

VAR

temp : position;

BEGIN

temp := p.next;

new(p.next);

p.next.element := x;

p.next.next := temp

END;

PROCEDURE Delete (p : position);

BEGIN

p.next := p.next.next;

END;

FUNCTION Locate (x : elementtype; L : LIST) : position;

VAR

p : position;

BEGIN

p := L;

WHILE p.next <> nil DO

IF p.next.element = x THEN

RETURN(p.next)

ELSE p := p.next;

RETURN(p.next) /\* if not found \*/

END;

§2.2 队列

一．什么是队列？

队列是一种特殊的表，它从尾部加入元素，从头部取出元素，满足先进先出的原则。

二．队列的操作：

1. Makenull(Q): 将队列Q置成空队列。

2. Front(Q): 返回队列头元素。

3. Enqueue(x, Q): 在队列Q的尾部加入元素x。

4. Dequeue(Q): 在队列Q的头部取出一个元素。

5. Empty(Q): 判断队列Q是否空队列。

三．队列的实现

1. 用链表实现

图2.2

2. 用循环数组实现

图2.3

四．队列的几个操作的算法

假设用循环数组实现队列

PROCEDURE Enqueue (x : elementtype; Q : QUEUE);

BEGIN

Q[tail(Q)] := x;

IF tail(Q) = length(Q) THEN

tail(Q) := 1

ELSE tail(Q) := tail(Q) + 1;

END;

PROCEDURE Dequeue (Q : QUEUE);

BEGIN

X := Q[head(Q)];

IF head(Q) = length(Q) THEN

Head(Q) := 1

ELSE head(Q) := head(Q) + 1;

RETURN(X)

END;

§2.3 栈

一．什么是栈？

栈是一种特殊的表，它在栈顶插入和删除元素，满足后进先出的原则。

二．栈的操作

1. Makenull(S): 将栈S置为空栈。

2. Top(S): 返回栈S的栈顶元素。

3. Pop(S): 删除S的栈顶元素。

4. Push(x, S): 在S的栈顶插入元素x。

5. Empty(S): 判断S是否空栈。

三．用数组实现栈

图2.4

三．栈的几个操作的实现

FUNCTION Empty (S : STACK) : boolean;

BEGIN

IF top(S) = 0 THEN

RETURN(true)

ELSE RETURN(false)

END;

PROCEDURE Push (x : elementtype; S : STACK);

BEGIN

top(S) := top(S) + 1;

S[top(S)] := x;

END;

PROCEDURE Pop (S : STACK);

BEGIN

IF Empty(S) THEN

Error(“underflow”)

ELSE BEGIN

top(S) := top(S) 1;

RETURN(S[top(S)+1]);

END

END;

四．栈的性质

进栈、出栈序列

图2.5

定理2.1：有可能利用一个栈从得到排列当且仅当不存在下标，使得。

证明：仅当）设是利用栈从得到的序列，假如存在下标使得。那么由栈的性质，在之前入栈，在之后之前入栈，由的排列顺序，在入栈之前，和均不出栈。入栈后，比先出栈，此时，由于比后入栈，因而比先出栈。故不可能得到这样的出栈序列。故不可能用栈从得到。

当）假设不存在下标使得。利用归纳法证明可用栈从得到。

当时，。显然利用栈可从1得到。

设时，结论也成立。

当时，设，并设是使的最小下标。当时，由归纳假设，可用栈从得到。最后进栈并出栈作为。从而得到。当时，存在。首先我们证明，恰好为所有小于r的数, (这时，)。假若不然，存在使得并且，这与本定理的条件矛盾。由归纳假设，可用栈从得到。当出栈后，入栈放在栈底，恰好为所有大于的数。由归纳假设，可用栈从得到。最后出栈，作为排在最后。从而可用栈从得到。

§2.4 树

一．有根树

1. 有根树的定义：

一棵树可以递归定义如下：

1. 一个结点自身是一棵树，它也是该树的根结点。
2. 假设是一个结点，是树，它们的根结点分别为。我们把作为的父结点，构造一棵新树。在这棵树中，是根结点，是根结点的子树。称为的孩子。

2. 例子：

图2.6

二．二叉树

1. 定义：每个结点至多有两个孩子的有根树称为二叉树。二叉树的每个结点的孩子分为左孩子和右孩子。

2. 用二叉树表示一般树

对每一棵有根树，可以构造一棵二叉树表示它。具体方法如下：的根结点对应的根结点。若中结点对应中结点，则的左孩子是在中的长子，的右孩子是在中的兄弟。

例如：图2.6可表示成图2.7.

图2.7

三．二叉树的遍历

1. 前序遍历

PROCEDURE Preorder (T : TREE);

BEGIN

IF T <> nil THEN

BEGIN

WRITE(T);

Preorder(T);

Preorder(T);

END

END;

2. 中序遍历

PROCEDURE Inorder (T : TREE);

BEGIN

IF T <> nil THEN

BEGIN

Inorder(T);

WRITE(T);

Inorder(T);

END

END;

中序遍历的非递归算法：

PROCEDURE Inorder (T : TREE);

/\*中序遍历以T为根的二叉树，S为栈\*/

BEGIN

Makenull(S);

REPEAT

WHILE T <> nil DO

BEGIN

Push(T, S);

T := Tlchild;

END;

IF NOT Empty(S) THEN

BEGIN

T := Pop(S);

WRITE(T.data);

T := T.rchild;

END;

UNTIL (T=nil) AND Empty(S);

END;

3. 后序遍历

PROCEDURE Postorder (T : TREE);

BEGIN

IF T <> nil THEN

BEGIN

Postorder(T);

Postorder(T);

WRITE(T);

END

END;

§2.5 图

1. 图

1.图的定义：一个图，其中是有穷、非空的顶点集，为边集。如果边是有序的顶点对，则图称为有向图。称为该边的尾，称为该边的头。如果边是无序的顶点对，则图称为无向图。

图2.8

2. 几个概念

在有向图中，如果是中的边，则我们说与邻接，我们也称是从到的边。和称为与边关联。从发出的边的数目称为的出度，进入的边的数目称为的入度，与关联的边的数目称为的度数

在无向图中，如果是中的边，则说与邻接，和称为与边关联。与关联的边数称为的度数。

一条有向(无向)路径是一系列边，其中是从到的边，。则称该路径是从到长度为的路径，用表示。

一条路径称为是简单的，是说该路径上的所有顶点和边互不相同。一个起点和终点相同的简单路径称为一个圈。

3. 图的表示方法

邻接矩阵：图2.8中, 两个图的邻接矩阵表示如图2.9中的和

图2.9

邻接表：图2.8中, 两个图的邻接表见图2.10中的和。

图2.10

二．树

一个图称为是连通的，是说：该图中任意两点和之间有一条路径相连。

一个连通无圈图称为一棵树。

例：

图2.11

这里的树是图论中的树，与有根树不同。