第四章 栈和队列

1.设将整数 1，2，3，4 依次进栈，但只要出栈时栈非空，则可将出栈操作按任何次序夹入其中，请回答下述问题：

(1)若入、出栈次序为 Push(1), Pop(),Push(2),Push(3), Pop(), Pop( ),Push(4), Pop( ),则出栈的数字序列为何(这里 Push(i)表示 i 进栈，Pop( )表示出栈)?

(2)能否得到出栈序列 1423 和 1432?并说明为什么不能得到或者如何得到。

(3)请分析 1，2 ，3 ，4 的 24 种排列中，哪些序列是可以通过相应的入出栈操作得到的。

答：(1)出栈序列为：1324

(2)不能得到 1423 序列。因为要得到 14 的出栈序列，则应做 Push(1),Pop(),Push(2),Push(3),Push(4),Pop()。这样，3 在栈顶，2 在栈底，所以不能得到 23 的出栈序列。能得到 1432 的出栈序列。具体操作为：Push(1), Pop(),Push(2),Push(3),Push(4),Pop(),Pop(),Pop()。

(3)在 1，2 ，3 ，4 的 24 种排列中，可通过相应入出栈操作得到的序列是：

1234,1243,1324,1342,1432,2134,2143,2314,2341,2431,3214,3241,3421,4321

不能得到的序列是：1423,2413,3124,3142,3412,4123,4132,4213,4231,4312

2.循环队列的优点是什么?如何判断它的空和满?

优点：相对于直线队列来讲，直线队列在元素出队后，头指针向后移动，导致删除元素后的空间无法在利用，即使元素个数小于空间大小，依然无法再进行插入，即所谓的“假上溢”。当变成循环队列之后，删除元素后的空间仍然可以利用，最大限度的利用空间。

判断方式：

1、采用计数器来判断，空时，计数器为 0，满时，计数器为 maxsize。

2、另设一个布尔变量以匹别队列的空和满。

3、少用一个元素的空间，约定入队前，测试尾指针在循环意义下加 1 后是否等于头指针，若相等则认为队满。

扩展资料：

三种基本运算：

1、置队空 voidInitQueue(CirQueue*Q){ Q->front=Q->rear=0;Q->count=0; }//计数器置 0。

2、判队空 intQueueEmpty(CirQueue*Q){ returnQ->count==0; }//队列无元素为空。

3、判队满 intQueueFull(CirQueue*Q){ returnQ->count==QueueSize;}//队中元素个数等于 QueueSize 时队满。

注意：队列的操作特点是“先进先出”。前者主要是头指针、尾指针的使用，后者主要是理解循环队列提出的原因及其特点。两者都要掌握队列空与满的判定条件以及出队列、入队列操作的实现。

5.回文是指正读反读均相同的字符序列，如“abba”和“abdba”均是回文，但“good”不是回文。试写一个算法判定给定的字符序列是否是回文。（提示：将一半的字符入栈）

```
int HuiWen(char *str)
{
    SqStack S;
    InitStact(S);
    int len = strlen(str);
    int i,judge=1;
    for(i = 1;i<=len/2;i++)
    {
        push(S,*str);
          *str++;
    }
    if(len%2 == 1)
           *str++;
```

```
    for(int j=i;j>0;j--)
    {
        pop(S,e);
        if(*str == e)
            *str++;
        else
            judge = 0;
    }
    return judge;
}
```

6.利用栈的基本操作，写一个将栈 S 中所有结点均删去的算法 void ClearStack (SeqStack *S），并说明 S 为何要作为指针参数?

7.设计算法判断一个算术表达式的圆括号是否正确配对。（提示：对表达式进行扫描，凡遇到’(’就进栈，遇’)’就退掉栈顶的’(’，表达式被扫描完毕，栈应为空。

```
#include <stdio.h>
#include <stdlib.h>
#define max 100
typedef struct sta
{
    char data[max];
    int top;
} stack;
stack* init()//栈初始化
{
    stack *s=(stack*)malloc(sizeof(stack));
    s->top=-1;
    return s;
}
int empty(stack *s)//判断是否空栈
{
    if(s->top==-1)
        return 1;
    else return 0;
}
int push(stack* s,char x)//压栈
{
    if(s->top==max-1) return -1;//若栈已满，则返回
    else
    {
        s->top++;
        s->data[s->top]=x;
        return 1;
    }
}
int pop(stack* s,char *x)//出栈
{
    if(empty(s)) return 0;
    else
    {
        *x=s->data[s->top];
        s->top--;
        return 1;
    }
}
```

```
int result()
{
	char x;int flag=1,flag1=1;//看是否能合适的入栈和出栈
	stack *s=init();
	//scanf("%c",&x);
	//getchar();//消除输入一个换行字符的影响，
	//while((x!='\n')zhe
	//这里会导致超时,x!='\n'这个条件一直都满足，跳不出循环，因为输入文件末尾根本就没有回车符，而在 codeblock 中输入完数据后会手动敲一个回车符，一个陷入死循环一个能正常结束，EOF 是文件结束标志
	while((x=getchar())!=EOF)//如果是'('则压栈，')'则出栈
	{
		if(x=='(')
			flag=push(s,x);
		if(x==')')
			flag1=pop(s,&x);
	}
    if(empty(s)&&flag==1&&flag1==1) return 1;//遍历完后栈中无括号则匹配正确
    else return -1;
}
int main()
{
	int flag;
	flag=result();
	if(flag==1)
		printf("1");
	else printf("0");
	return 0;
}
```

8.一个双向栈 S 是在同一向量空间内实现的两个栈，它们的栈底分别设在向量空间的两端。试为此双向栈设计初始化 InitStack（S）、入栈 Push（S，i，x） 和出栈 Pop（S，i）等算法，其中 i 为 0 或 1，用以表示栈号。

9.

10.对于循环向量中的循环队列，写出求队列长度的公式。

11.假设循环队列中只设 rear 和 quelen 来分别指示队尾元素的位置和队中元素的个数，试给出判别此循环队列的队满条件，并写出相应的入队和出队算法，要求出队时需返回队头元素。

```
#define QueueSize 100
typedef char Datatype ; //设元素的类型为 char 型
typedef struct {
	 int quelen;
	 int rear;
	 Datatype Data[QueueSize];
    }CirQueue;
  CirQueue *Q;
```

循环队列的队满条件是：

```
	Q->quelen==QueueSize
```

(1)判断队满

```
int FullQueue( CirQueue *Q)
{   //判队满,队中元素个数等于空间大小
    return Q->quelen==QueueSize;
        }
```

(2)入队

```
void EnQueue( CirQueue *Q, Datatype x)
{   //  入队
```

```
if(FullQueue( Q))
      Error(“队已满，无法入队”);
  Q->rear=(Q->rear+1)%QueueSize;
     Q->Data[Q->rear]=x;
Q->quelen++;   }
```

(3)出队

```
Datatype DeQueue( CirQueue *Q)
{ //出队
if(Q->quelen==0)
      Error("队已空，无元素可出队");
int tmpfront;   //设一个临时队头指针
tmpfront=(QueueSize+Q->rear - Q->quelen+1)%QueueSize
Q->quelen--;
   return Q->Data[tmpfront];            }
```

第五章 矩阵和广义表

1．设 A[0..9，0..9]是一个 10×10 对称矩阵，采用压缩存储方式存储其下三角部分，已知每个元素占用两个存储单元，其第一个元素 A 0，0 的存储位置为 1000，求如下问题的计算过程及结果：

1）给出 A 4，5 的存储位置；

2）给出存储位置为 1080 的元素下标。

2．设计算法，计算一个三元组表表示的稀疏矩阵的对角线元素之和。

```
#include "stdio.h"
typedef struct
{ int row;
int col;
int data;
}Triple;
int MDSum(Triple *a)
{ int i;
int sum=0;
if (a[0].row!=a[0].col)
return ERROR;
for (i=1;i<=a[0].data;i++)
{ if (a[i].row==a[i].col)
sum+=a[i].data;
}
return sum;
}
```

3．设有三对角矩阵 An*n，将其三条对角线上的元素逐行地存储到向量 B[0...3n-3]中，使得 B[k]=aij,求：

(1)用 i , j 表示 k 的下标变换公式。

要求 i,j 到 k 的下标变换公式，就是要知道在 k 之前已有几个非零元素，这些非零元素的个数就是 k 的值，一个元素所在行为 i,所在列为 j,则在其前面已有的非零元素个数为：

```
(i*3-1)+j-(i+1)
```

其中 (i*3-1)是这个元素前面所有行的非零元素个数，j-(i+1)是它所在列前面的非零元素个数

化简可得：

```
k=2i+j; // c 下标是从 0 开始的。
```

(2)用 k 表示 i, j 的下标变换公式。

因为 K 和 i,j 是一一对应的关系，因此这也不难算出：

```
i=(k+1)/3 //k+1 表示当前元素前有几个非零元素,被 3 整除就得到行号

j=(k+1)% 3+(k+1)/3-1 //k+1 除以 3 的余数就是表示当前行中第几个非零元素，

//加上前面的 0 元素所点列数就是当前列号
```

4．设二维数组 A5*6 的每个元素占 4 个字节，已知 Loc(a00)=1000，A 共占多少个字节？A 的终端结点 a45 的起始地位为何?按行和按列优先存储时，a25 的起始地址分别为何？

字节占 5x6x4=120 字节　a45 地址是 1116　 a25 地址是 1068

5．特殊矩阵和稀疏矩阵哪一种压缩存储后会失去随机存取的功能?为什么？

稀疏矩阵压缩存储后，必会失去随机存取功能。稀疏矩阵在采用压缩存储后将会失去随机存储的功能。因为在这种矩阵中，非零元素的分布是没有规律的，为了压缩存储，就将每一个非零元素的值和它所在的行、列号做为一个结点存放在一起，这样的结点组成的线性表中叫三元组表，它已不是简单的向量，所以无法用下标直接存取矩阵中的元素。

第六章 二叉树和树

1．给出只有 3 个结点的所有二叉树。

2．已知二叉树有 50 个叶子结点，则该二叉树的总结点数至少应该有多少个？

1． 二叉树共用 3 类结点,即度为 2 的结点,度为 1 的结点和度为 0 的结点（叶子结点）；

2、任何一个二叉树的叶子结点数总比度为 2 的结点数多一个；

3、至少的情况就是该二叉树为满二叉树,及没有度为 1 的结点； 故,50+49=99.

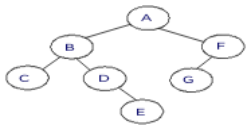
3．具有 n 个结点的满二叉树的叶子结点的个数是多少？

n 个结点的满二叉树叶子结点数为（n + 1) / 2

推导过程为：由二叉树的属性可知 N2 = N0 – 1。由于满二叉树没有度为 1 的结点，所以 N0 + N2 = n

因此 2 × N0 – 1 = n。N0 = （n + 1) / 2

4、已知二叉树的先序、中序和后序序列分别如下，其中有一些看不清的字母用*表示，请构造出一棵符合条件的二叉树并画出该二叉树。先序序列是：*BCFG 中序序列是：CB*EAG* 后序序列是：*EDB*FA**



5、假设二叉树采用二叉链表存储结构存储，设计一个算法，求先序遍历序列中第 k(1≤k≤二叉树结点个数)个结点的值。

```
BiTNode    searchk(BiTree T,int k)
{ if ( T ) {
            count++;       //  全局变量，对结点计数
            if (count==k)    return    T;
 searchk( T->lchild, k);
 searchk(T->rchild, k);
  } // if
    return NULL;            }
```

6．以二叉链表为存储结构，分别写出求二叉树结点总数及叶子总数的算法。

7．以二叉链表为存储结构， 写一算法交换各结点的左右子树。

```
void ChangeBinTree(BiTree    T)      {     //交换子树
   if(T) {     //后序遍历
     BiTNode    *temp;
   ChangeBinTree(T->lchild);
   ChangeBinTree(T->rchild);
   temp=T->lchild;
   T->lchild=T->rchild;
   T->rchild=temp;             }
}
```

8．以二叉链表为存储结构，分别写出在二叉树中查找值为 x 的结点及求 x 所在结点在树中层数的算法。

```
//求层次
int lnLevel(BiTree T,DataType x){
int static L=0; //设一静态变量保存层数
if(T)      {
      if(L==0) //若是访问根结点
        {
          L++;   //第 1 层
   if(T->data==x)    return L ;
   if(T->lchild||T->rchild)
     L++; //若根有子树，则层数加 1
                }
else {    //访问子树结点
if(T->data==x)    return L ;
if(T->lchild||T->rchild)
          L++;   //若该结点有子树，则层数加 1
       else    return 0;
     }
   lnLevel(T->lchild,x); //遍历左子树
lnLevel(T->rchild,x); //遍历右子树
   }   // if(T)
```

```
}
```

9. 一棵 n 个结点的完全二叉树以向量作为存储结构，试写一非递归算法实现对该树的前序遍历。

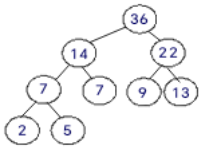
```
/*以向量为存储结构的完全二叉树，其存储在向量中的结点其实是按层次遍历的次序存放的*/
//设结点数据类型为 char
typedef char DataType;
#define M 100 //设结点数不超过 100
typedef DataType BinTree[M];
void Preorder(BinTree T)
{ //前序遍历算法
   int n=T[0];   //树中结点数
   int p[M]; //设置一队列存放结点值
   int i,j;
   for(i=1;i<=n;i++)  {
      if (i==1)  //根结点
            j=1;
      else   if(2*j<=n)   //左子树
            j=2*j;
      else if(j%2==0&&j<n)   //右兄弟
                  j=j+1;
             else if(j>1)   //双亲之右兄弟
                  j=j/2+1;
   p[i]=T[j]; //入队
   printf("%c",p[i]);   //打印结点值
}
}
```

10. 以线索链表作为存储结构。分别写出在前序线索树中查找给定结点*p 的前序后继，以及在后序线索树中查找 *p 的后序前趋的算法。

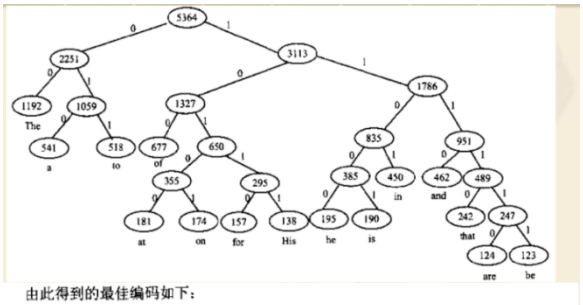
```
BinThrNode * SearchPostInPre(BinThrNode *p){
      //查找结点*p 的前序后继
   if(p){
      if(p->rtag==0)&&(p->ltag==0)
            /*当左、右都为孩子指针，p 的前序后继为左孩子*/
         return p->lchild;
   else return p->rchild;   }
}

BinThrNode *SearchPreInPost(BinThrNode *p)
        /*当有左线索或无有孩子, p 的后续前趋为 p->lchild */
        return p->lchild;
   else return p->rchild;   }
}
```

11. 以数据集{2，5，7，9，13}为权值构造一棵哈夫曼树，并计算其带权路径长度。



12.如下表给出了在一篇有 19710 个词的英文文章中出现最普通的 15 个单词的出现频度。假定一篇正文仅由上述字符数据表中的词组成，那么它们的最佳编码是什么？平均长度是多少？



由此得到的最佳编码如下：

The:	00	of:	100	a:	010	to:	011	and:	1110
in:	1101	that:	11110	he:	11000	is:	11001	at:	10100
on:	10101	for:	10110	His:	10111	are:	111110	be:	111111

平均长度=(1192*2+677*3+541*3+518*3+462*3+450*4+242*5+195*5+
190*5+181*5+174*5+157*5+138*5+124*6+123*6)/5364=3.48

13、已知一棵度为 m 的树中有 n1 个度为 1 的结点，n2 个度为 2 的结点，...nm 个度为 m 的结点，问该树中有多少片叶子?

14、高度为 h 的完全二叉树至少有多少个结点?至多有多少个结点?

15、试找出分别满足下面条件的所有二叉树：

(1)前序序列和中序序列相同；　　(2)中序序列和后序序列相同；

(3)前序序列和后序序列相同；　　(4)前序、中序、后序序列均相同。

15、高度为 h 的严格二叉树至少有多少个结点?至多有多少个结点?

16、在什么样的情况下，等长编码是最优的前缀码?

第七章 图

1．试在无向图的邻接矩阵和邻接链表上实现如下算法：

（1)往图中插入一个顶点　(2)往图中插入一条边　(3)删去图中某顶点　(4)删去图中某条边

```
#define MAX_VERTEX_NUM   30
typedef num{DG,DN,UDG,UDN}GraphKind;
typedef   ArcType AdjMtrix
[MAX_VERTEX_NUM][MAX_VERTEX_NUM];
typedef struct {
        VertexType vex[MAX_VERTEX_NUM];
        AdjMtrix arc;
        int vexnum,arcnum;
     GraphKind kind;}GraphMtrix;
(1)往图中插入一个顶点
 AddVertex(GraphMtrix *G,VertexType x)
{ / /往无向图的邻接矩阵中插入顶点
if (G->vexnum>= MAX_VERTEX_NUM )
        Error("顶点数太多");
else   {G->vex[G-> vexnum]=x;
         //将新顶点输入顶点表
     G->vexnum++;  //顶点数加 1
//邻接矩阵增加第 n+1 行，第 n+1 列
for(k=0;k<=G->vexnum;k++)
   G.arc[k][vexnum]=G.arc[vexnum][k]=0;
  }    //else
}
(2)  往图中插入一条边
 Addedge (GraphMtrix *G,VertexType x,VertexType y)
{   //往无向图的邻接矩阵中插入边(x,y)
   int i,j,k;
   i=-1;j=-1;
   for(k=0;k<G-vexnumn;k++)
            //查找 X，Y 的编号
     { if (G->vexs[k]===x) i=k;
       if (G->vexs[k]==y) j=k;          }

if (i==-1||j==-1) Error("结点不存在");
        else {   //插入边(x,y)
                 G->arc[i][j]=1;
                 G->arc[j][i]=1;
                 G->vexarc++;   //边数加 1
                 }
     }

(3)删去图中某顶点
void DeleteVertexMGraph(MGraph *G,VertexType x)
{   //无向图的邻接矩阵中删除顶点 x
   int i,k,j;
   i=-1;
   for(k=0;k<G->vexnum;k++)
          //查找 X 的编号
     if (G->vex[k]==x)    i=k;
   if (i==-1) Error("结点不存在");
   else
{   //删除顶点以及边
   k=0; //求出与 x 结点相关联的边数 k
```

```
   for (j=0;j<G->vexnum;j++)
      if (G->vex[i][j]==1)    k++;
                  //设置新的边数
   G->arcnum=G->arcnum-k;
   //在邻接矩阵中删除第 i 行
      for (k=i+1;k<G->vexnum;k++)
         for(j=0;j<G->vexnum;j++)
     G->arc[k-1][j]=G->arc[k][j];
   //在邻接矩阵中删除第 i 列
for (k=i+1;k<G->vexnum;k++)
   for(j=0;j<G->vexnum;j++)
         G->arc[j][k-1]=G->vexs[j][k];
G->vexnum--;   //总结点数-1
   }   //else
}
```

(4)删去图中某条边

```
void DeleteedgeMGraph(MGraph *G,VertexType x,VertexType y)
{     //无向图的邻接矩阵中删除边(x,y)
   int i,j,k;
   i=-1;j=-1;
        //查找 X，Y 的编号
   for(k=0;k<G->vexnum;k++)
      { if (G->vex[k]==x) i=k;
         if (G->vex[k]=y)    j=k;          }

if (i==-1||j==-1) Error("结点不存在");
else if (g->vexs[i][j]==1)
{   //删除边(x,y)
   G->arc[i][j]=0;
   G->arc[j][i]=0;
   G->arcnum--;   //边数减 1
}   // else
}
```

2．试以邻接表和邻接矩阵为存储结构，分别写出基于 DFS 和 BFS 遍历的算法来判别顶点 vi 和 vj(i<>j)之间是否有路径。

3．试分别写出求 DFS 和 BFS 生成树(或生成森林)的算法，要求打印出所有的树边。

4．写一算法求连通分量的个数并输出各连通分量的顶点集。

```
   typedef enum{FALSE，TRUE}Boolean；//FALSE 为 0，TRUE 为 1
   Boolean visited[MaxVertexNum]；  //访问标志向量是全局量
   void DFSTraverse(ALGraph *G)
```

```
{ //深度优先遍历以邻接表表示的图 G，求连通分量的个数和各连通分量的顶点集
  int i;
  for(i=0;i<G->n;i++)
    visited[i]=FALSE;  //标志向量初始化
  j=0;//连通分量个数计数器
  for(i=0;i<G->n；i++)
    if(!visited[i]) //vi 未访问过
      {j++;
        printf("connected component %d:{",j);
        DFS(G，i);  //以 vi 为源点开始 DFS 搜索
        printf("}\n");
      }
}//DFSTraverse
void DFS(ALGraph *G，int i){
  //以 vi 为出发点对邻接表表示的图 G 进行深度优先搜索
  EdgeNode *p;
  printf("%c,"，G->adjlist[i].vertex)；//访问顶点 vi
  visited[i]=TRUE;  //标记 vi 已访问
  p=G->adjlist[i].firstedge;  //取 vi 边表的头指针
  while(p){//依次搜索 vi 的邻接点 vj，这里 j=p->adjvex
      if  (!visited[p->adjvex])//若 vi 尚未被访问
        DFS(g，p->adjvex);//则以 Vj 为出发点向纵深搜索
      p=p->next;  //找 vi 的下一邻接点
    }
}//DFS
```

5．设图中各边的权值都相等，试以邻接矩阵和邻接表为存储结构，分别写出算法：

(1)求顶点 vi 到顶点 vj(i<>j)的最短路径

(2)求源点 vi 到其余各顶点的最短路径　要求输出路径上的所有顶点(提示：利用 BFS 遍历的思想)

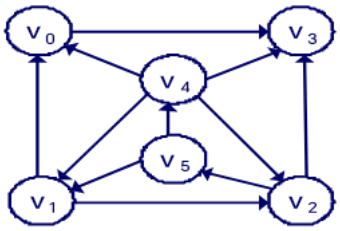
6．以邻接表为存储结构，写一个基于 DFS 遍历策略的算法，求图中通过某顶点 vk 的简单回路(若存在)。

```
int circleDFS(ALGraph *G，int k){
  //以 vk 为出发点对邻接表表示的图 G,求简单回路,若存在返回 1，否则返回 0
  EdgeNode *p;
  printf("%c"，G->adjlist[k].vertex)；//访问顶点 vk
  visited[k]=TRUE;  //标记 vk 已访问
  p=G->adjlist[k].firstedge;  //取 vk 边表的头指针
  while(p){//依次搜索 vk 的邻接点 vj，这里 j=p->adjvex
      if (!visited[p->adjvex])//若 vj 尚未被访问
          DFS(G，p->adjvex);//则以 Vj 为出发点向纵深搜索
      else if (p->adjvex==k)
              {printf("%c",G->adjlist[k].vertex)；return 1;}
      p=p->next;  //找 vk 的下一邻接点
    }
  return 0;
}//DFS
```

7．设有向图 G 如下图 1 所示，试给出：

（1）该图的邻接矩阵；（2）该图的邻接表；（3）该图的逆邻接表；（4）从 V0 出发的“深度优先”遍历序列；　(5)

从 V0 出发的“广度优先”遍历序列；　(6) 画出从 V0 出发的深度优先生成树和广度优先生成树。



8．解答下面的问题：

(1) 求网的最小生成树有哪些算法？各适用何种情况？为什么？

(2) 有如下的网络邻接矩阵，画出一棵最小生成树。

$$\begin{bmatrix} \infty & 17 & \infty & \infty & 20 & 22 \\ 17 & \infty & 6 & 7 & \infty & 12 \\ \infty & 6 & \infty & 11 & \infty & \infty \\ \infty & 7 & 11 & \infty & 19 & 15 \\ 20 & \infty & \infty & 19 & \infty & 34 \\ 22 & 12 & \infty & 15 & 34 & \infty \end{bmatrix}$$

9．有一带权无向图的顶点集合为{v0，v1，v2，v3，v4，v5，v6，v7}。已知其邻接矩阵的三元组表示如下所示。

(1) 画出该无向图的邻接表。　(2) 画出所有可能的最小生成树。

(3) 根据 (1) 所得的邻接表分别写出从 v0 出发进行深度优先遍历和广度优先遍历序列。

(4) 求出从 v0 到其余个顶点的最短路经。　(5) 求出各顶点对之间的最短路经。

```
8    8    20
0    1    12
0    4     2
1    0    12
1    5     3
1    7     5
2    3     8
2    4     2
2    5     4
3    2     8
3    4    10
3    6     8
4    0     2
4    2     2
4    3    10
5    1     3
5    2     4
5    6     7
6    3     8
6    5     7
7    1     5
```

第八章 查找

1．若对具有 n 个元素的有序的顺序表和无序的顺序表分别进行顺序查找，试在下述两种情况下分别讨论两者在等概率时的平均查找长度：

（1）查找不成功，即表中无关键字等于给定值 K 的记录；

（2）查找成功，即表中有关键字等于给定值 K 的记录。

查找不成功时，需进行 n+1 次比较才能确定查找失败。因此平均查找长度为 n+1 ，这时有序表和无序 表是一样的。

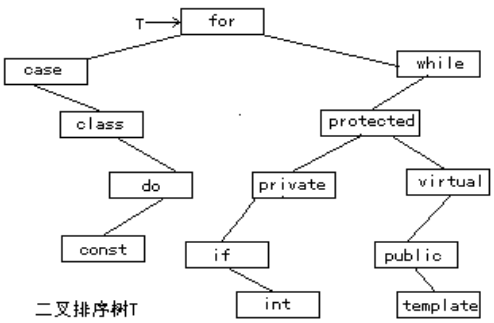
查找成功时，平均查找长度为 (n+1)/2, 有序表和无序表也是一样的。因为顺序查找与表的初始序列状态 无关。

2．试分别画出在线性表（a,b,c,d,e,f,g）中进行折半查找，以查关键字等于 e,f 和 h 的过程。

3．画出对长度为 18 的有序的顺序表进行二分查找的判定树，并指出在等概率时查找成功的平均查找长度，以及查找失败时所需的最多的关键字比较次数。

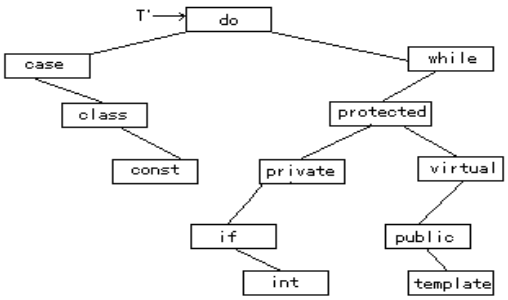
4．将(for, case, while, class, protected, virtual, public, private, do, template, const ,if, int)中的关键字依次插入初态为空的二叉排序树中，请画出所得到的树 T。然后画出删去 for 之后的二叉排序树 T',若再将 for 插入 T'中得到的二叉排序树 T"是否与 T 相同?最后给出 T" 的先序、中序和后序序列。

二叉排序树 T 如下图：

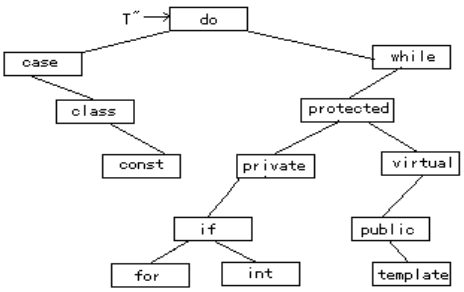


二叉排序树T

删去 for 后的二叉排序树如下图：



再插入结点 for 后的二叉排序树 T":



5．对给定的关键字集合，以不同的次序插入初始为空的树中，是否有可能得到同一棵二叉排序树?

6．将二叉排序树 T 的先序序列中的关键字依次插入一空树中，所得和二叉排序树 T'与 T 否相同?为什么?

7．设二叉排序树中关键字由 1 至 1000 的整数构成，现要查找关键字为 363 的结点，下述关键字序列哪一个不可能是在二叉排序树上查找到的序列

(a) 2，252，401，398，330, 344，397，363；

(b) 924, 220, 911, 244, 898, 258, 362, 363;

(c) 925, 202, 911, 240, 912, 245, 363;

(d) 2, 399, 387, 219, 266, 382, 381, 278, 363.

散列结构

1、散列存储与其它存储主要有什么区别?

2、发生冲突都有哪些解决冲突的办法?

3、已知关键字序列为(PAL,LAP,PAM,MAP,PAT,PET,SET,SAT,TAT,BAT)试为它们设计一个散列函数，将其映射到区间[0..n-1]上，要求碰撞尽可能的少。这里 n=11,13,17,19.

4、对于一组给定的、固定不变的关键字序列，有可能设计出无冲突的散列函数 H，此时称 H 为完备的散列函数(perfect hashing function)，若 H 能无冲突地将关键字完全填满散列表，则称 H 是最小完备(minimal perfect)的散列函数。通常找完备的散列函数非常困难，找最小完备的散列函数就更困难。请问:
（1）若 h 是已知关键字集合 K 的完备的散列函数，若要增加一个新的关键字到集合 K，一般情况下 H 还是完备的吗?

(3) 已知关键字集合为(81，129，301，38，434，216，412，487，234)，散列函数为 H(x)=(x+18)/63,请问 H 是完备的吗?它是最小完备的吗?

(3)考虑由字符串构成的关键字集合(Bret, Jane, Shirley, Bryce, Michelle, Heather)，试为散列表[0..6]，设计一个完备的散列函数。(提示：考虑每个字符串的第 3 个字符，即 s[2])

6、设散列表长度为 11，散列函数 h(x)=x%11,给定的关键字序列为：1，13，13，34，38，33，27，22.试画出分别用拉链法和线性探查法解决冲突时所构造的散列表，并求出在等概率情况下，这两种方法查找成功和失败时的平均查找长度。请问装填因子的值是什么?

排　序

1．比较直接插入排序和希尔排序的不同点。

2．给出关键字序列{4，5，1，2，8，6，7，3，10，9}的直接插入排序过程。

3．给出关键字序列{4，5，1，2，8，6，7，3，10，9}的希尔排序过程。

4．指出堆和二叉排序树的区别。

5．给出关键字序列{4，5，1，2，8，6，7，3，10，9}的冒泡排序过程。

6．设计算法，实现双向冒泡排序算法，即在排序过程中交替改变扫描方向。

7．给出关键字序列{4，5，1，2，8，6，7，3，10，9}的二路归并排序过程。

8．给出关键字序列{4，5，1，2，8，6，7，3，10，9}的基数排序过程。

9．将哨兵放在 R[n]中，被排序的记录放在 R[0..n-1]中，重写直接插入排序算法。

10．以单链表作为存储结构实现直接插入排序算法。

11．设计算法，使得在尽可能少的时间内重排数组，将所有取负值的关键字放在所有取非负值的关键字之前。请分析算法的时间复杂度。

12．下面是一个自上往下扫描的冒泡排序的伪代码算法，它采用 lastExchange 来记录每趟扫描中进行交换的最后一个元素的位置，并以它作为下一趟排序循环终止的控制值。请仿照它写一个自下往上扫描的冒泡排序算法。

```
void BubbleSort(int A[],int n)
  //不妨设 A[0..n-1]是整型向量
  int lastExchange,j,i=n-1;
while (i>0)
     lastExchange=0;
     for(j=0;j<i;j++)//从上往下扫描 A[0..i]
          if(A[j+1]<A[j]){
               交换 A[j]和 A[j+1];
               lastExchange=j;
          }
     i=lastExchange;//将 i 置为最后交换的位置   }//endwhile   }//BubbleSort
```

13．改写快速排序算法，要求采用三者取中的方式选择划分的基准记录；若当前被排序的区间长度小于等于 3 时，无须划分而是直接采用直接插入方式对其排序。

14. 对给定的 j(1≤j≤n),要求在无序的记录区 R[1..n]中找到按关键字自小到大排在第 j 个位置上的记录(即在无序集合中找到第 j 个最小元)，试利用快速排序的划分思想编写算法实现上述的查找操作。

15．以单链表为存储结构，写一个直接选择排序算法。

16．写一个建堆算法：从空堆开始，依次读入元素调用上题中堆插入算法将其插入堆中。

17．已知两个单链表中的元素递增有序，试写算法将这两个有序表归并成一个递增有序的单链表。算法应利用原有的链表结点空间。果可输出到另一个向量 B[0..n-1]中。

18．设向量 A[0..n-1]中存有 n 个互不相同的整数，且每个元素的值均在 0 到 n-1 之间。试写一时间为 O(n)的算法将向量 A 排序，结果可输出到另一个向量 B[0..n-1]中。