# 习题2 表结构

2-1 如果长度为n的线性表采用顺序存储结构存储，则在第i (1≤i≤n+1)个位置插入一个新元素的算法的时间复杂度为( B)。

(A) O(1)

(B) O(n)

(C) O(nlog2n)

(D) O(n2)

2-2 在一个有127个元素的顺序表中插入一个新元素，要求保持顺序表元素的原有(相对)顺序不变，则平均要移动( C)个元素。

(A) 7

(B) 32

(C) 64

(D) 127

2-3 已知An×n为稀疏矩阵。试从时间和空间角度比较，采用二维数组和三元组顺序表两种存储结构计算∑aij的优缺点。

答:

二维数组

优点:直接将整个稀疏矩阵存储在连续内存中，访问起来很快，元素排列直观。

缺点:稀疏矩阵中很多元素都是0，花费大量空间存储0元素，造成空间浪费。

三元顺序表

优点:只存储非零元素和它们所在的行列下标，空间利用率高，节省空间。

缺点:访问元素不像二维数组直接通过下标访问，还需要查找操作，时间复杂度会更高一些。

2-4 综合比较顺序表和链表。

存储:

顺序表存储空间利用率高，只用来存储元素值，而且采用顺序存储结构。内存需要预先分配，固定容量，可能会造成内存浪费。

链表存储空间利用率低，除了存储数据信息还需要存储指针，采用链式存储结构。内存动态分配，需要增添元素再开辟新节点。

元素访问:

顺序表访问元素需要的时间少，通过下标直接访问对应元素。时间复杂度O(1)。

链表需要访问元素时往往需要对链表进行遍历，时间复杂度O(n)。

插入删除:

顺序表插入删除元素需要移动其他元素，操作比较麻烦，时间开销大。时间复杂度O(n)。

链表插入删除元素只需要需对目标元素所在节点的相邻节点进行处理，时间短。时间复杂度O(1)。

2-5 解释链表的“头指针、头结点和首元素结点”三个概念。

**头指针**指的是线性链表中第一个结点或头结点的存储地址，它是访问链表的起始点。

**头节点**指的是附加在第一个数据元素之前的结点，该结点的数据域一般为“空”、指针域存放第一个数据元素的地址。

**首元素**节点指的是第一个数据域存放数据元素的结点，头节点指向首元素节点。

2-6 设链表L→a→b→c→d，指针域为\*next。执行下列命令后，( B )。

p=L->next->next;

L->next->next=NULL;

q=L->next->next;

(A) p→b→c→d，q→a

(B) p→b→c→d，q→NULL

(C) p→c→d，q→a

(D) p→c→d，q→a→b

2-7 描述下列算法的主要功能是( A )。

① 构造头结点L，取q=L;

② 产生1个结点p;

③ q−>next=p;

④ 输入p−>data的值;

⑤ 取q=p;

⑥ 重复执行②至⑤n次;

⑦ p−>next=NULL;

(A) 通过输入n个数据元素构建链表L

(B) 采用前插法，在链表L中输入n个数据元素

(C) 通过产生n个结点构建链栈L，q为栈顶指针

(D) 在链队列L中输入n个数据元素，q为队尾指针

2-8 设两个循环链表的长度分别为n和m，则将这两个循环链表连接成一个循环链表，最好的时间复杂度为( A )。

(A) O(1)

(B) O(n)

(C) O(m)

(D) O(min(n，m))

2-9 设push和pop分别表示进栈和出栈操作，输入序列为xyz，则经过栈操作(  A)可以输出序列yzx。

(A) push, push, pop, push, pop, pop

(B) push, push, push, pop, pop, pop

(C) push, pop, push, pop, push, pop

(D) push, pop, push, push, pop, pop

2-10 设进栈序列为123，试给出所有可能的出栈序列。

1. 3、2、1
2. 2、3、1
3. 2、1、3
4. 1、3、2
5. 1、2、3

2-11 如果进栈序列为123456，能否得到出栈序列435612和135426?

答:两个序列都可以得到

2-12 简述算法的功能(设数据元素类型为int)：

void proc(LinkQueue \*Q)

{

LinkStack S;

InitStack(S);

while(!EmptyQueue(Q) )

{

DeleteQueue(Q, d);

Push(S,d);

}

while(!EmptyStack(S) )

{

Pop(S, d);

InsertQueue(Q, d);

}

}

这个算法的功能是将链队列中的元素按顺序出列并压入链栈中，然后再将链栈中的元素按顺序出栈，重新入列，实现了链队列的元素逆序排列。

2-13 描述下列递归算法的功能。

int F(int m, int n)

{

if (n>m) return F(n, m);

else if (n==0) return m;

else return F(n, m%n);

}

该算法是GCD算法，用于计算m,n两数的最大公约数，若m比n小，则进入F(n,m)递归，保证函数第一个参数总是大于第二个参数的，当n不为0时，由辗转相除法可以知道有F(m.n)=F(n,m%n)恒成立，而当第二个参数n为0 时，此时的第一个参数m即为传入的m，n的最大公约数。

2-14 编写递归算法：

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  | 0，m=0且n≥0 |
|  | g(m, n)= |  |
|  |  | g(m-1, 2n)+n，m>0且n≥0 |
|  |  |  |  |

int F(int m, int n)

{

if (m == 0 && n >= 0)return 0;

else if(m>0&&n>=0)return F(m - 1, 2 \* n) + n;

}

2-15 将下列递归过程改写为非递归过程。

void test(int &s)

{

int x;

scanf ("%d", &x);

if (x==0) s=0;

else

{

test(s);

s+=x;

}

}

该函数的作用是计算输入的整数之和，当输入0时表示输入停止。

非递归:  
void test(int& s)

{

int x;

do {

scanf("% d", & x);

s += x;

}while(x!=0)

}

2-16 按照格式要求给出调用F(3,'A','B','C')的运行结果：

void F(int n, char x, char y, char z)

{

if (n==1) printf("1 %c 🡪 %c\n", x, z);

else

{

F(n-1, x, z, y);

printf("%d %c 🡪 %c\n", n, x, z);

F(n-1, y, x, z);

}

}

输出结果：

