《数据结构与算法》作业 2

22920212204392 黄勖

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($nlog_2n$)
(D) O(n^2)

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：设稀疏矩阵为 m 行 n 列,如果采用二维数组存储,其空间复杂度为 O(m×n);因为要将所有的矩阵元素累加起来,所以,需要用一个两层的嵌套循环,其时间复杂度亦为 O(m× n) 。如果采用三元组顺序表进行压缩存储,假设矩阵中有 t 个非零元素,其空间复杂度为 O(t),将所有的矩阵元素累加起来只需将三元组顺序表扫描一遍,其时间复杂度亦为 O (t) 。当 t<< m× n 时,采用三元组顺序表存储可获得较好的时、空性能。

2-4 综合比较顺序表和链表。

答：1.基于空间的考虑

顺序表的存储空间是静态分配的，在程序执行之前必须声明它的规模。若线性表的长度变化较大，则存储规模难以预先确定，过大会造成空间浪费，过小可能会导致溢出。动态链表的存储空间是动态分配的，只要内存有空闲，就不会导致溢出。因此，当线性表的长度变化较大，难以估计其存储规模时，采用动态链表作为存储结构比较好。

存储密度是指结点数据本身所占的存储量和整个结点结构所占的存储量之比。一般来说，存储密度越大，存储空间的利用率就越高。显然顺序表的存储密度为 1，而链表的存储密度小于 1。例如，单链表中各结点的数据均为整数，指针所占空间和整型量相同，则单链表的存储密度为 0.5。因此若不考虑顺序表中的备用结点空间，则顺序表的空间利用率是 1。单链表的空间利用率是 0.5。因此，当线性表的长度变化不大，易于事先确定其大小时，为了节约存储空间，采用顺序表作为存储结构比较好。

2.基于时间的考虑

顺序表是一种随机存储结构,对表中任一结点都可以在 O(1)时间内直接存

取，而链表中的结点则需要从头指针开始顺序遍历才能取得。因此，若线性表的操作主要是进行查找，很少做插入或删除时，采用顺序表为宜。

在链表的任意位置上进行插入删除，都只需要修改指针。而在顺序表中，平均要移动一半的结点【(0+1+2+3+…+n)/(n+1)=n/2】。因此，对于频繁进行插入删除操作的线性表，采用链表为宜。若表的插入删除操作主要发生在表的首尾两端，则采用带尾指针的单循环链表。

3.基于语言的考虑

在没有提供指针类型的高级语言环境中，若要采用链表结构，则可以使用游标来实现静态链表。虽然静态链表在存储分配上有不足之处，但它和动态链表一样，具有插入删除方便的特点。

2-5 解释链表的“头指针、头结点和首元素结点”三个概念。

答：“头指针”是指向头结点的指针；

“头结点” 是为了操作的统一、方便而设立的，放在首元素结点之前，其数据域一般无意义；

“首元结点”也就是第一元素结点，它是头结点后边的第一个结点。

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。

答：所有可能的出栈序列为：

1,2,3

1,3,2

2,1,3

2,3,1

3,2,1

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

答：不能得到 435612。6 的后面不可能将 1,2 出栈
135426 能够得到。

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);
    }
}
```

答：将链式队列 (LinkQueue) 中的数字顺序倒过来

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);
}
```

答：求 m 与 n 的最大公约数

2-14 编写递归算法：

$$g(m,n)=\begin{cases}0, & m=0 \text{ 且 } n\geq 0\\ g(m-1,2n)+n, & m>0 \text{ 且 } n\geq 0\end{cases}$$

答：

```
double g(double m,double n){
    if(m==0&&n>=0)
        return 0;
    else
        return g(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;
    }
}
```

答：

```
void test(int &s){
    int x;
    s = 0;
    while(cin >> x && x!=0){
        s+=x;
    }
}
```

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);
        }
    }
```

答:

1 A→C

2 A→B

1 C→B

3 A→C

1 B→A

2 B→C

1 A→C