# 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：**空间上：对稀疏矩阵A求和时，用二维数组存储的主要优点就是更加直观，以及可以通过下标直接取出对应的元素进行操作，缺点也很明显，稀疏矩阵中有大量的0，为了存储所有非零元素，需要填充整个数组，意味着很多空间会花在零值上，导致空间占用大。

而用三元组顺序表存储，优点是只要存储需要的元素及其对应下标，节省了大量的空间，缺点是不够直观。

时间上：二维数组为了计算所有非零元素的和，需要遍历整个矩阵。对于稀疏矩阵，意味着很多时间会花在遍历零值上，导致效率低下。而三元组顺序表因为只存储了非零元素，所以遍历整个列表即可得到所有非零元素的和，无需考虑零值。

2-4 综合比较顺序表和链表。

**答：**1.时间上：

顺序表：支持随机访问，任何元素的访问时间为O(1)，但插入和删除操作需要移动大量元素，时间复杂度为O(n)。

链表：不支持随机访问，查找特定元素需要遍历链表，最坏时间复杂度为O(n)，但插入和删除操作只需调整指针，时间复杂度为O(1)。

2.空间上：

顺序表：需要预先分配固定大小的空间，适合固定大小的数据集，避免空间浪费或溢出。

链表：动态分配空间，适合大小不定或频繁变化的数据集，但每个节点需额外存储指针信息，导致空间利用率低于顺序表。

2-5 解释链表的“头指针、头结点和首元素结点”三个概念。

**答：（**1）头指针：头指针是指向链表中第一个结点的指针。若链表存在头结点，则头指针指向头结点；若链表没有头结点，则头指针指向首元素结点。

（2）头结点：头结点是链表中首元素结点之前的一个节点，头结点的指针域指向首元素结点，头结点的数据域可以不存储信息，也可以存储附加信息，如链表长度等。

（3）首元素结点：首元素结点是链表中存储第一个数据元素的结点。

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：(1)321 (2)231 (3)213 (4)132 (5)123**

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，可以得到135426。**

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：**利用堆栈将队列Q中的数据元素逆置。

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：

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

int g (int m, int n) {

    if (m == 0 && n >= 0)

        return 0;

    else if (m > 0 && n >= 0)

        return g (m-1, 2\*n) + n;

    return 0;

}

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;

}

}

void add (int &s) {

    int x;

    s = 0;

    scanf ("%d", &x);

    while (x != 0) {

        s += x;

        scanf ("%d", &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);

}

}

运行结果如下：

1 A->C

2 A->B

1 C->B

3 A->C

1 B->A

2 B->C

1 A->B