# 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：

**时间角度：**

优点：对于随机访问矩阵中的元素非常方便，能够直接通过下标快速定位到特定的元素 ，计算总和时可以直接遍历二维数组，对于规模较小的稀疏矩阵，时间效率相对较高。

缺点：如果稀疏矩阵规模较大，遍历整个二维数组会比较耗时，因为大部分元素可能都是零，而对零元素的遍历浪费了大量时间。

**空间角度：**

优点：直观易懂，编程实现相对简单。

缺点：对于稀疏矩阵来说，会浪费大量的存储空间来存储零元素。

**三元组顺序表：**

**时间角度：**

优点：对于非常稀疏的矩阵，在计算总和时，只需要遍历存储非零元素的三元组，避免了对大量零元素的处理，时间效率较高。特别是当矩阵规模很大且稀疏度很高时，优势更加明显。

缺点：由于三元组顺序表不是随机存储结构，不能像二维数组那样通过下标直接定位元素，在访问特定元素时可能需要遍历整个三元组顺序表，对于一些需要频繁随机访问元素的操作，时间开销较大。

**空间角度：**

优点：只存储非零元素及其行、列下标，大大节省了存储空间，尤其对于稀疏度很高的矩阵，能够极大地减少存储需求。

缺点：编程实现相对复杂一些，需要额外的代码来处理三元组的存储和操作。并且在进行一些操作时，可能需要动态调整三元组顺序表的大小，增加了一定的时间开销。

2-4 综合比较顺序表和链表。

答：**一、存储结构**

**顺序表：**采用一段连续的存储单元依次存储线性表的数据元素。可以通过下标直接访问任意元素，随机访问速度快。

**链表：**由一系列节点组成，每个节点包含数据域和指针域。节点之间通过指针链接，存储位置不一定连续。无法像顺序表那样通过下标直接访问元素，需要从链表的头节点开始逐个遍历节点才能找到特定元素。

**二、存储分配方式**

**顺序表：**存储分配方式分为静态分配和动态分配。静态分配在编译时确定大小，一旦空间用完不能扩充；动态分配可以在运行时根据需求动态调整大小，但可能涉及到数据的搬迁，操作相对复杂。

**链表：**只要内存空间还有剩余，就可以随时分配新的节点，无需预先确定大小，存储分配更加灵活。

**三、插入和删除操作**

**顺序表：**在插入和删除元素时，需要移动大量元素以保持元素的连续性。如果在表头或表中间进行插入或删除操作，时间复杂度为 O (n)，其中 n 是顺序表的长度。

**链表：**插入和删除操作只需修改相关节点的指针即可，时间复杂度为 O (1)。但如果要查找插入或删除的位置，可能需要遍历链表，时间复杂度为 O (n)。

**四、空间利用率**

**顺序表：**预先分配连续的存储空间，如果存储的元素较少，可能会造成空间的浪费；但如果存储的元素较多且分配的空间刚好合适，空间利用率较高。

**链表：**每个节点除了存储数据外，还需要额外的指针域空间，因此空间利用率相对较低。但对于存储不连续的数据或大小不确定的数据，链表可以更好地利用内存空间。

**五、适用场景**

**顺序表：**适合频繁进行随机访问、元素数量相对稳定、不需要频繁插入和删除操作的场景。例如，存储一组固定大小的学生成绩数据，可快速通过下标访问特定学生的成绩。

**链表：**适合频繁进行插入和删除操作、元素数量动态变化较大的场景。例如，实现一个动态的任务队列，任务可以随时添加或删除。

2-5 解释链表的“头指针、头结点和首元素结点”三个概念。

答：**一、头指针**

头指针是指向链表中第一个节点的指针。它的作用十分重要：用于标识链表的起始位置。通过头指针可以快速访问链表中的第一个节点，进而遍历整个链表。如果头指针为 null，则表示链表为空。

例如，在单链表中，头指针通常被命名为 head，可以通过 head->next 来访问链表的第二个节点，以此类推，实现对整个链表的遍历操作。

**二、头结点**

头结点是为了方便链表操作而在链表的首元素结点之前额外创建的一个节点。它具有以下特点：（1）头结点的数据域一般不存储具体的数据信息（也可以根据需求存储一些特殊信息，比如链表长度等）。（2）头结点的指针域指向首元素结点。（3）引入头结点的好处在于可以使链表的插入和删除操作在逻辑上更加统一，避免了对空链表和非空链表进行不同处理的情况。例如，在进行插入操作时，无论链表是否为空，都可以直接在头结点之后进行插入，无需特殊判断链表是否为空。

**三、首元素结点**

首元素结点是链表中存储第一个有效数据元素的节点。它具有以下特点：（1）数据域存储了具体的元素值。（2）指针域指向下一个节点（如果是单链表）或者指向前一个节点和下一个节点（如果是双向链表）。（3）首元素结点是链表中真正存储用户所需数据的第一个节点，通过遍历链表可以依次访问到后续的各个元素结点。

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( C)。

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( D)。

(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：123，132，213，231，321。

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

答：前者无法得到，后者可以得到。

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);

}

}

答：该算法的功能是将一个链队列中的元素逆序。

具体过程如下：

1. 首先初始化一个链栈S。
2. 然后，当链队列Q不为空时，不断从队列中取出元素d，并将其压入栈S中。这个过程实际上是把队列中的元素依次转移到了栈中。由于栈具有 “后进先出” 的特性，此时元素在栈中的顺序与在队列中的顺序相反。
3. 接着，当栈S不为空时，不断从栈中弹出元素d，并将其插入到队列Q中。这个过程是把逆序后的元素重新放回队列中，从而实现了队列中元素的逆序操作。

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：上述代码是用递归方式实现了求两个整数的最大公约数的函数。首先判断 n > m 的情况：如果 n 大于 m，则调用 F(n, m)，这样就确保了第一个参数始终是较大的数，第二个参数始终是较小的数。接着判断 n == 0 的情况：如果 n 等于 0，则返回 m。根据最大公约数的定义，当一个数为 0 时，另一个数就是它们的最大公约数。最后是一般情况：此时返回 F(n, m%n)。这是基于欧几里得算法（辗转相除法）的原理，两个整数的最大公约数等于较小数和两数相除余数的最大公约数。不断地进行这样的递归操作，直到 n 为 0，此时就找到了最大公约数并返回。

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 |
|  |  |  |  |

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：#include <stdio.h>

void test(int &s)

{

int x;

int temp\_s = 0;

while (1)

{

scanf("%d", &x);

if (x == 0) { s = temp\_s; break; }

else { temp\_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 B C  
 3 A C  
 1 B A  
 2 B C  
 1 A C