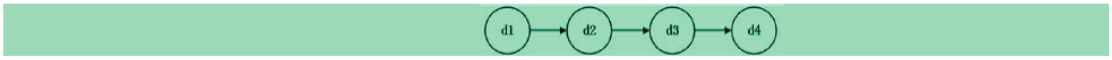
1 CHP1. 绪论

1. 设有数据结构（D，R），其中
D={d1,d2,d3,d4}，R={r}，r={(d1,d2),(d2,d3),(d3,d4)}。
请画出其对应逻辑结构。

1.3❷设有数据结构（D，R），其中

D={d1,d2,d3,d4}，R={r}，r={(d1,d2),(d2,d3),(d3,d4)}。

试按图论中图的画法惯例画出其逻辑结构图。



2. 下列程序段的时间复杂度是（　　）。

```
i=1; k=0;
while(i<=n-1){
      k+=10*I;
      i++;
}
```

A．$O(\log_{10}n)$　　B．O(n)　　C．不确定　　D．$O(n^2)$

解：循环执行 n-1 次，大 O 记法写为$O(n)$，应选 **B**

3. 下列程序段的时间复杂度是（　　）。

```
k=0;
for(i=1;i<=n;i++)
    for(j=1;j<=i;j++)
        for(k=1;k<=j;k++)
              x+=1;
```

A．$O(n^3)$　　B．O(n)　　C．不确定　　D．$O(n^2)$

解：
最深层循环执行次数为:

$$\sum_{i=1}^{n}\sum_{j=1}^{i}\sum_{k=1}^{j}1=\sum_{i=1}^{n}\sum_{j=1}^{i}j=\sum_{i=1}^{n}(1+i)\times i/2=\left(\sum_{i=1}^{n}i+\sum_{i=1}^{n}i^2\right)/2$$
$$=((1+n)\times n/2+n(1+n)(1+2n)/6)/2=n(1+n)(2+n)/6$$

因此时间复杂度应为 $O(n^3)$

$$1^2+2^2+3^2+\ldots\ldots+n^2=n(n+1)(2n+1)/6$$

应选 **A**

2 CHP2. 线性表

4. 填空

(1) 在顺序表中插入或删除一个元素，平均约需要移动_____元素，具体移动的元素个数与_____有关。

(2) 顺序表中逻辑上相邻的元素的物理位置_____紧邻。单链表中逻辑上相邻的元素的物理位置_____紧邻。

(3) 在单链表中，除了首元结点外，任意结点的存储位置由_____指示。

(4) 在单链表中设置头结点的作用是_____

解：

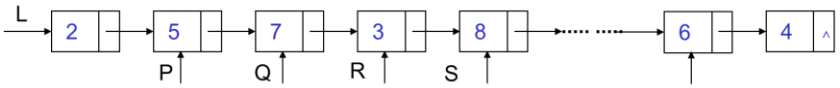
a) 在顺序表中插入或删除一个元素，平均约需要移动 表中一半 元素，具体移动的元素个数与 表长和该元素在表中的位置 有关。

b) 顺序表中逻辑上相邻的元素的物理位置 必然 紧邻。单链表中逻辑上相邻的元素的物理位置 未必 紧邻。

c) 在单链表中，除了首元结点外，任意结点的存储位置由 其直接前驱的指针域 指示。

d) 在单链表中设置头结点的作用是 在表头进行插入或删除的操作与其他位置的操作相同（插入或删除首元素时不必进行特殊处理）

5. 对以下单链表分别执行下列各程序段，并画出结果示意图。



(1) Q=P->next

(2) L=P->next

(3) R->data=P->data

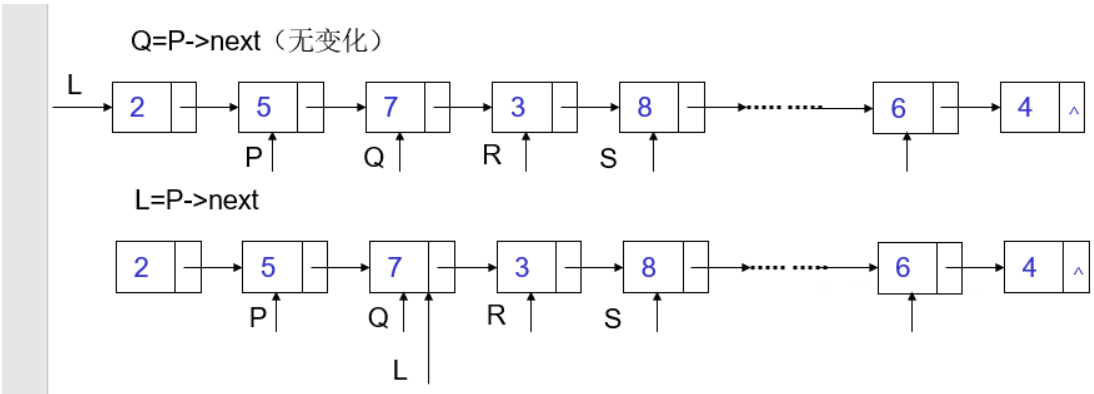
(4) R->data=P->next->data

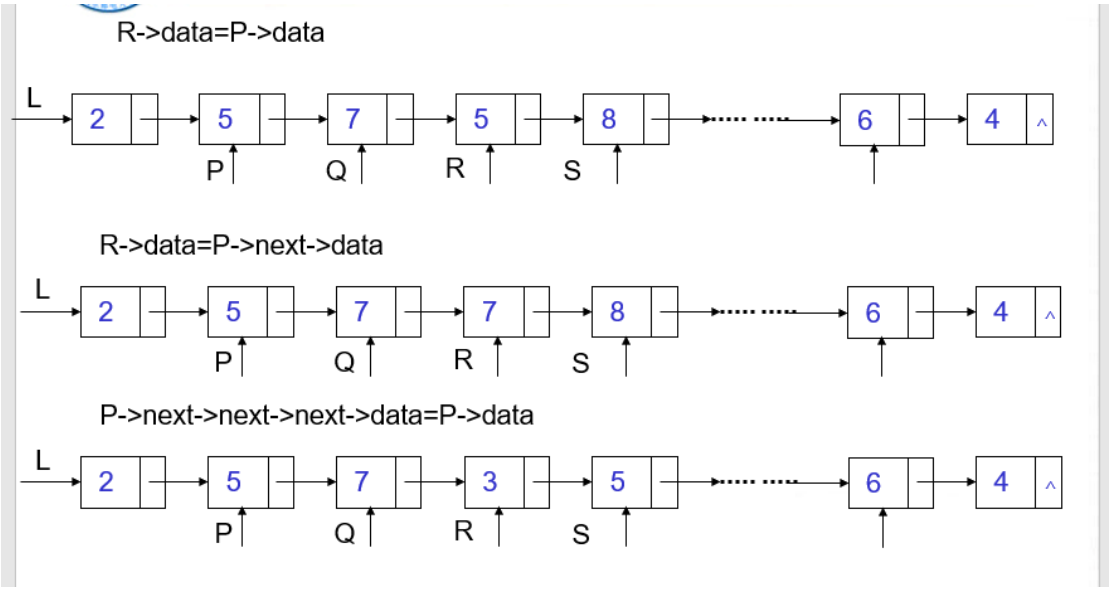
(5) P->next->next->next->data=P->data

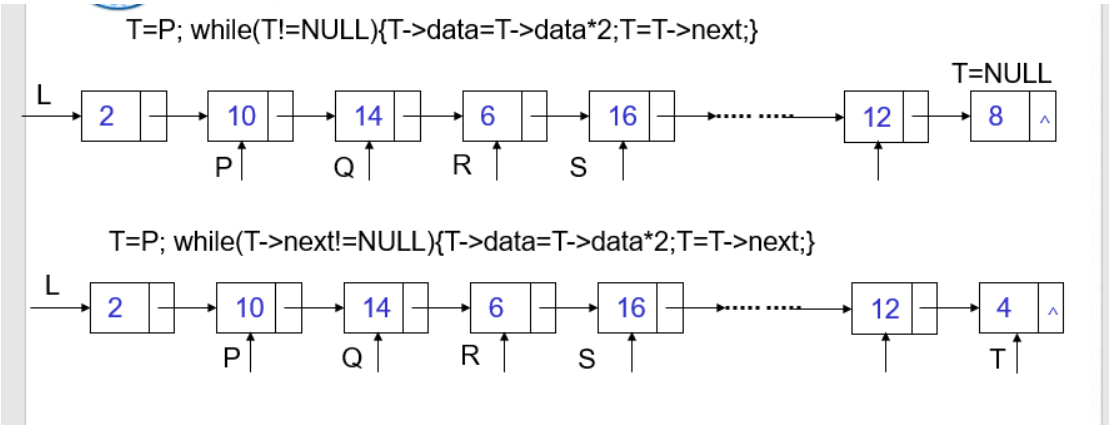
(6) T=P; while(T!=NULL){T->data=T->data*2;T=T->next;}

(7) T=P; while(T->next!=NULL){T->data=T->data*2;T=T->next;}

解：







6. 已知 L 是带表头结点的非空链表，且 P 结点既不是首元素结点，也不是尾元素结点，试从下列提供的答案中算则合适的语句序列。

(1) 删除 P 结点的直接后继结点的语句序列是_____

(2) 删除 P 结点的直接前驱结点的语句序列是_____

(3) 删除 P 结点的语句序列是_____

(4) 删除首元结点的语句序列是_____

(5) 删除尾元结点的语句序列是_____

语句有：

(1) P=P->next;

(2) P->next=P;

(3) P->next=P->next->next;

(4) P=P->next->next;

(5) while(P!=NULL) P=P->next;

(6) while(Q->next!=NULL) {P=Q; Q=Q->next;}

(7) while(P->next!=Q) P=P->next;

(8) while(P->next->next!=Q)P=P->next;

(9) while(P->next->next!=NULL) P=P->next;

(10) Q=P;
(11) Q=P->next;
(12) P=L;
(13) L=L->next;
(14) free(Q);

解：

a) 删除 P 结点的直接后继结点的语句序列是 (11)(3)(14)
b) 删除 P 结点的直接前驱结点的语句序列是 (10)(12)(8)(11)(3)(14)
c) 删除 P 结点的语句序列是 (10)(12)(7)(3)(14)
d) 删除首元结点的语句序列是 (12) (11)(3)(14)
e) 删除尾元结点的语句序列是 (9)(11)(3)(14)

7. 试写一算法，对单链表实现就地逆置。

解：

```
typedef struct node
{
    int data;
    struct node *next;
}NODE, *PNODE, *LinkList;
void Invert(LinkList head) {//逆置链表,链表已经存在，head指向其头结点。
    NODE *p = NULL,
        *q = head->next;//head指向头结点，q指向a1

    head->next = NULL;//头结点从原链表中断开
    while (q)
    {
        p = q;//摘出结点ai
        q = q->next;
        p->next = head->next;
        head->next = p;//将ai插入表头，形成逆序
    }
}
```

8. 请分析在有序单链表中删除一个结点并保持有序的时间复杂度。如果单链表无序呢？

解：O(n)

主要操作为比较待删除元素和当前结点元素值，删除第 i 个结点需要比较 i 次，因此最少比较 1 次，最多比较 n 次，平均比较(n+1)/2（基于等概率假设：各个结点被删的概率相等）。

单链表无序时，删除任一结点之前都需要遍历整个链表，遍历 n 个结点的时间复杂度为 O(n)

9. “访问线性表的第 i 个元素的时间必然同 i 有关”，这个说法正确吗？为什么？

解：
不正确。若线性表采用顺序存储，则访问元素的时间复杂度为 O(1)，与 i 无关。若采用链式存储，则指针移动的次数与元素位置 i 有关。

10. 请分析循环单链表表示的链队列中，是否可以不设置队头指针？
解：
可以。循环链表中，通过指针域，整个链表形成一个环，从任意结点出发都能找到表中其他结点。所以，循环单链表中，保留指向表头的指针就可以了。
队列采用循环单链表表示时，即使不设队头指针，也可正常完成插入、删除、访问等操作。

11. 比较线性表顺序存储和链式存储的特色。对比这两种存储结构下插入操作的时间复杂度。
解：
（1）顺序存储
优点：存储密度大，存储空间利用率高。随机存取，便于查询。
缺点：插入或删除元素时不方便。
（2）链式存储
优点：插入或删除元素时很方便，使用灵活。
缺点：存储密度小，存储空间利用率低。不能随机存取，不便于查询。
（3）比较
顺序表适宜于做查找这样的静态操作
链表宜于做插入、删除这样的动态操作

12. 已知线性表中的元素是无序的，且以带头结点的单链表 L 作为存储结构。设计一个删除表中所有值小于 a（a 为给定数值）的元素的算法，并给出算法中关键步骤的注释说明。
解：

```
typedef struct Node{
   ElemType data;
   struct Node *next;
}Node, *PNode;

typedef struct LinkList{
   int count;//总数
   PNode head;
}LinkList;

void DelElem(PNode head, ElemType a){
   PNode q = head, p = head->next;

   while(p ! = NULL){
      if(p->data < a){
         q->next = p->next;
```

```
            free(p);
            p=q->next;
        }else{
            q = p; p = p->next;
        }
    }
}
```

3 CHP3. 栈和队列

13. 若按教科书 3.1.1 节中图 3.1(b)所示铁道进行车厢调度（注意：两侧铁道均为单向行驶道），则请回答：

(1) 如果进站的车厢序列为 123，则可能得到的出站车厢序列是什么？

(2) 如果进站的车厢序列为 123456，则能否得到 435612 和 135426 的出站序列，并请为什么不能得到或者如何得到？（即写出以‘S’表示进栈和‘X’表示出栈的栈操作序列）。

解：

(1) 123,132,213,231,321,

123 对应的操作序列为：SXSXSX；132 对应的操作序列为：SXSSXX；213 对应的操作序列为：SSXXSX；231 对应的操作序列为：SSXSXX；321 对应的操作序列为：SSSXXX；

(2) 不能得到 435612，因为，首先出站的车厢为 4 意味着 123 号车厢已经入栈，则这两节车厢出站的相对顺序一定是 3,2,1，而该序列中，1 号车厢在 2 号车厢之前出站，这不符合操作规则。

能得到 135426，操作序列为：SXSSXSSXXXSX

14. 分析下面程序段的功能。

```
status Del(Stack S,int e)
{
  Stack T; int d;
  InitStack(T);
  while(!StackEmpty(S))
  {
    Pop(S, d);
    if(d!=e)
    Push(T, d);
  }
  while(!StackEmpty(T))
  {
    Pop(T, d);
    Push(S, d);
  }
```

}

解：如果栈 S 种存在元素 e，将其删除。

请将下面代码以非递归方式改写。

```
void test(int &sum)
{
   int x;
   cin>>x;
   if(x==0)
      sum=0;
   else
   {
      test(sum);
      sum+=x;
   }
   cout<<sum;
}
```

解：

```
void test(int &sum){
   int x = 0, sum = 0;
   Stack s;

   InitStack(s);
   cin >> x
   while(x!=0){
      Push(s, x)
      cin >> x;
   }
   cout << sum;
   while(Pop(s,x)){
      sum += x;
      cout << sum;
   }
}
```

15. 写出以下程序段的输出结果（队列中的元素类型 QElemType 为 char）

```
void main(){
   Queue Q; Init Queue(Q);
   char x='e',y='c';
   EnQueue(Q,'h'); EnQueue(Q,'r');EnQueue(Q,y);
   DeQueue(Q,x);EnQueue(Q,x);
   DeQueue(Q,x);EnQueue(Q,'a');
   while(!QueueEmpty(Q)){
```

```
    DeQueue(Q,y);
    printf(y);
  }
  printf(x);
}
```

解：

EnQueue(Q,'h'); EnQueue(Q,'r');EnQueue(Q,y)执行后：Q(hrc)

DeQueue(Q,x);EnQueue(Q,x); 执行后：x='h',Q(rch)

DeQueue(Q,x);EnQueue(Q,'a'); 执行后：x='r',Q(cha)

while(!QueueEmpty(Q)){ DeQueue(Q,y); printf(y); }执行后：cha

printf(x); 执行后：char

16. 简述以下算法的功能（栈和队列的元素类型为 int）

```
void algo3(Queue &Q){
  Stack S; int d;
  InitStack(S);
  while(!QueueEmpty(Q)){
    DeQueue(Q, d); Push(S, d);
  }
  while(!StackEmpty(S)){
    Pop(S, d); EnQueue(Q, d);
  }
}
```

解：算法的功能是利用栈 S 作为辅助将队列 Q 中的元素倒置。

17. 如果希望循环队列中的元素都得到利用，则需要设置一个标志域，并以标志域真假来区分，尾指针和头指针相等时的队列状态是“空”还是“满”。试编写与此结构相应的队列基本操作。并从时间和空间角度来讨论设标志和不设标志这两种方法的使用范围。

解：

```
#define MAXQSIZE (3)
typedef int QElemType;
typedef struct _SqQueue {
    QElemType   *base;
    int         front;  //头指针，若队列不为空，指向队列头元素
    int         rear;   //尾指针，若队列不为空，指向队尾元素的下一个位置
    bool    empty;  //队列状态标识，true表示队列空
}SqQueue;

Status InitQueue(SqQueue *Q) {
    Q->base = (QElemType *)malloc(MAXQSIZE * sizeof(QElemType));
    if (!Q->base)
        return ERROR_OVERFLOW;
    Q->empty = true;
```

```
    Q->front = Q->rear = 0;
    return OK;
}

int QueueLength(SqQueue *Q) {
    int len = (Q->rear - Q->front + MAXQSIZE) % MAXQSIZE;
    if (len == 0 && !Q->empty){
        len = MAXQSIZE;
    }
    return len;
}

Status EnQueue(SqQueue *Q, QElemType e) {
    if (!Q->empty && Q->front == Q->rear)
        return ERROR_OVERFLOW;//队列满了，无法插入元素
    Q->base[Q->rear] = e;
    Q->rear = (Q->rear + 1) % MAXQSIZE;
    if (Q->rear == Q->front)
        Q->empty = false;//插入元素后，队列满了
    return OK;
}

Status DeQueue(SqQueue *Q, QElemType *e) {
    if (Q->empty && Q->front == Q->rear)
        return ERROR_EMPTY;//队列空了，无法删除元素
    *e = Q->base[Q->front];
    Q->front = (Q->front + 1) % MAXQSIZE;
    if (Q->rear == Q->front)
        Q->empty = true;//删除元素后，队列空了
    return OK;
}

Status DestroyQueue(SqQueue *Q) {
    if (!Q->base) return ERROR;
    free(Q->base);
    Q->front = Q->rear = -1;
    return OK;
}

Status PrintQueue(SqQueue *Q) {
    int pos = Q->front, len = QueueLength(Q), i = 0;

    printf("SqQueue:");
    for(i = 0; i < len; i++){
```

```
        printf("%d\t", Q->base[pos]);
        pos = (pos + 1) % MAXQSIZE;
    }
    printf("\n");
    return OK;
}
```

18. 栈和队列是操作受限的线性表，入栈和出栈操作在_____位置进行，队列的插入和删除操作是在_____位置进行。

解：栈顶，栈顶，队尾，队头

19. “栈和队列是非线性结构”说法正确吗？

解：错误。

20. 队列的顺序存储方式一般组织成为环状队列的形式，而且一般队列头或尾其中之一应该特殊处理。若队列头指针为 front，队列尾指针为 rear，请写出循环队列判空和判满方法。

解：

循环队列解决了用向量表示队列所出现的“假溢出”问题，但同时又出现了如何判断队列的满与空问题。

针对这个问题有两种方法。第一是增加标识位，指示队列是空还是满。

第二种是牺牲一个单元。即 front==rear 为队空，而（rear+1）%表长==front 为队满，这里表长为 n。

（front 指针指向队头元素，队头元素被删除后，头指针更新为 front=(front+1)%表长；rear 指向队尾新元素的插入位置，若队未满，先插入元素 array[rear]=e，再更新位置，rear = (rear+1)%表长）

4 CHP4. 串

21. 设 s=’I AM A STUDENT’, t= ‘GOOD’, q = ‘WORKER’。求：StrLength(s), StrLength(t), SubString(s,8,7), SubString(t,2,1), Index(s, ‘A’), Index(s, t), Replace(s, ‘STUDENT’,q), Concat(SubString(s,6,2), Concat(t, SubString(s, 7, 8)))。

解：

StrLength(s)为 14， StrLength(t)为 4， SubString(s,8,7)为’STUDENT’， SubString(t,2,1)为’O’，Index(s, ‘A’) 为 3， Index(s, t) 为 0, Replace(s, ‘STUDENT’,q) 为 ’I AM A WORKER’，Concat(SubString(s,6,2), Concat(t, SubString(s, 7, 8)))为‘A GOOD STUDENT’

22. 在以链表存储串值时，存储密度是结点大小和串长的函数。假设每个字符占一个字节，每个指针占 4 个字节，每个结点的大小为 4 的整数倍。求结点大小为 4k，串长为 l 时的存储密度 d(4k,l)

解：

每个结点存储 4k 个字符，串长为 l 需要 ceil(l/4(k-1))个结点（ceil 为向上取整函数），因此存储密度为：$l/(4(k+1)\times\lfloor l/4k\rfloor)$

23. 以定长顺序存储表示串，不允许调用串的基本操作，编写算法，从串 S 中删除所有和串 T 相等的子串。

解：

```
#define OK (0)
#define ERROR (1)
#define ERROR_OVERFLOW (2)
#define ERROR_EMPTY (3)
#define NULL (0)

#define MAXSTRLEN (255)
typedef unsigned char SString[MAXSTRLEN + 1];

void get_nextval(SString T, int nextval[]){
    int i = 1, j = 0;

    nextval[1] = 0;
    while (i < T[0]){
        if (j == 0 || T[i] == T[j]) {
            i++; j++;
            if (T[i] == T[j]){
                nextval[i] = nextval[j];
            }else{
                nextval[i] = j;
            }
        }else{
            j = nextval[j];
        }
    }
}

int Index_KMP(SString S, SString T, int nextval[], int pos) {
    int i = pos, j = 1;

    while (i <= S[0] && j <= T[0])
    {
        if (j == 0 || S[i] == T[j]) {
            i++; j++;
        }
        else {
            j = nextval[j];
        }
```

```
    }
    if (j > T[0])
        return i - T[0];
    else
        return 0;
}

int RemoveSubString(SString S, SString T) {
    int *nextval = (int *)malloc(sizeof(int) * (T[0]+1));
    int k = 1, i = 1, j = 0;

    if (nextval == NULL)
        return ERROR_OVERFLOW;
    memset(nextval, 0, sizeof(int) * (T[0] + 1));
    get_nextval(T, nextval);

    j = Index_KMP(S, T, nextval, i);
    while (j > 0 && i <= S[0] - T[0] + 1)
    {
        for (; i < j; ){//本次检索起始位置i，模式出现起始位置j，i到j-1之间的字符应该保留
            S[k++] = S[i++];
        }
        i += T[0];//下一次检索位置
        j = Index_KMP(S, T, nextval, i);
    }
    while (i <= S[0])
    {
        S[k++] = S[i++];
    }
    S[0] = k - 1;//更新长度
    free(nextval);
    return OK;
}
```