在包含1000个元素的线性表中实现如下各运算,所需执行时间最长的是 ( ) C

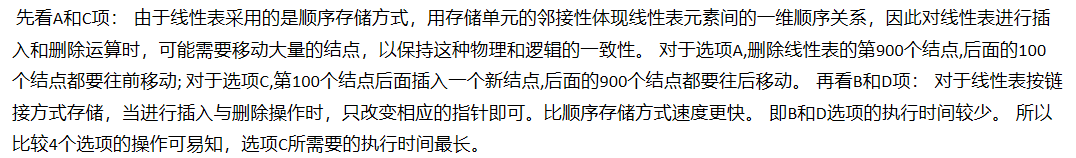
A.线性表按顺序方式存储，删除线性表的第900个结点

B.线性表按链式方式存储，删除指针P所指向的结点

C.线性表按顺序方式存储，在线性表的第100个结点后面插入一个新结点

D.线性表按链式方式存储，在线性表的第100个结点后面插入一个新结点

解析：





线性表的链式存储结构既方便其存取操作，也方便其插入与删除操作，这种说法（）A

A正确

B错误

B错误

墨迹绘图

以下关于单向链表说法正确的是（）？ABC


以下关于单向链表说法正确的是（）？ABC

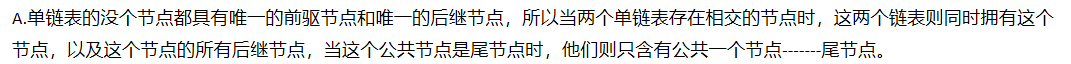
A.如果两个单向链表相交，那他们的尾结点一定相同

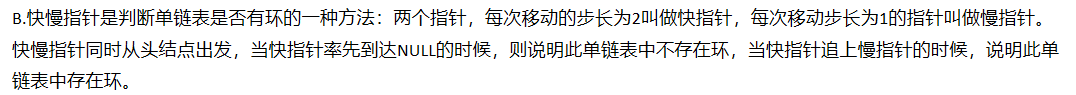
B.快慢指针是判断一个单向链表有没有环的一种方法

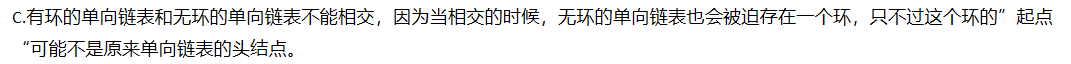
C.有环的单向链表跟无环的单向链表不可能相交

D.如果两个单向链表相交，那这两个链表都一定不存在环

解析：







C.有环的单向链表和无环的单向链表不能相交，因为当相交的时候，无环的单向链表也会被迫存在一个环，只不过这个环的”起点￼“可能不是原来单向链表的头结点。 
D.两个单向链表之间相交可以存在环。
墨迹绘图
判断下列说法是否正确：F=(a,F)是一个递归的广义表，它的深度是1,长度是2。( ）B
A.正确


A.正确

B.错误

B.错误
解析：广义表以及递归广义表的原理。 广义表是由n个元素组成的序列，n是广义表的长度。 广义表的深度： 广义表中括号的最大￼层数叫广义表的深度。 F=(a,F)的长度为2，由于属于递归表，所以深度为无穷，F相当于一个无限的表(a,(a,(a,(...))))。
墨迹绘图

已知广义表LS=((a,b,c),(d,e,f)),运用head和tail函数取出LS中原子e的运算是( )？C


已知广义表LS=((a,b,c),(d,e,f)),运用head和tail函数取出LS中原子e的运算是( )？C

A.head(tail(LS))

B.tail(head(LS))

C.head(tail(head(tail(LS)))

D.head(tail(tail(head(LS))))

解析：任何一个非空广义表的表头是表中第一个元素，它可以是原子，也可以是子表，而其表尾必定是子表。

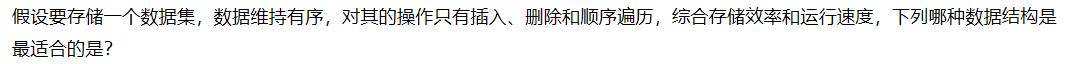
tail(LS)=((d,e,f))

head(tail(LS))=(d,e,f)

tail(head(tail(LS))) = (e,f)

tail(head(tail(LS))) = (e,f) 
head(tail(head(tail(LS)))) = e 
墨迹绘图

假设要存储一个数据集，数据维持有序，对其的操作只有插入、删除和顺序遍历，综合存储效率和运行速度，下列哪种数据结构是￼最适合的是？

A.数组

B.链表

C.哈希表

D.队列

解析：

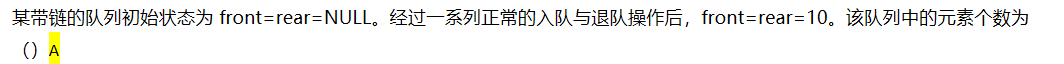
A.数组可以实现顺序遍历但是插入删除操作复杂，平均移动n/2个元素

B.链表因为存储的地址不连续（逻辑上连续实际上不连续），可以实现顺序遍历

C.哈希表是随机存储，所以是离散分布，顺序遍历实现不了 ,哈希表难以实现顺序遍历 队列插入删除效率低下.

D.队列只可以在队尾插入队头删除，不可以实现中间插入和删除，不满足条件



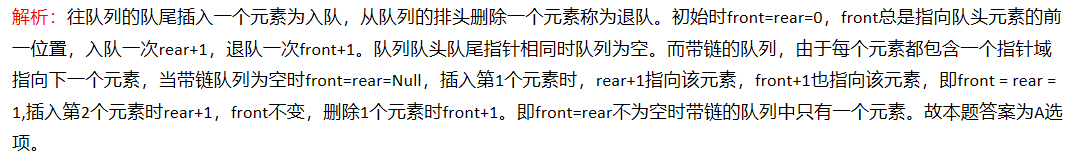


A.1

B.0

C.1或0

D.不确定





在一个长度为n的单链表的第i（0<=i<n）个元素后面插入一个元素时，需要向后移动（　  ）个元素。 D

A.n-i

B.n-i+1

C.n-i-1

D.0

解析：单链表在执行插入、删除时 不需要移动元素



链表不具有的特点是（）B

A.插入、删除不需要移动元素

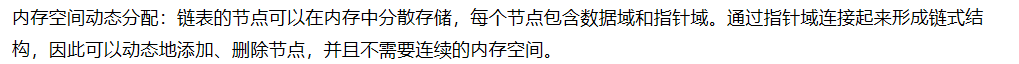
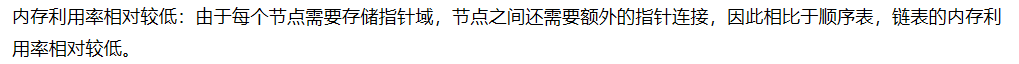
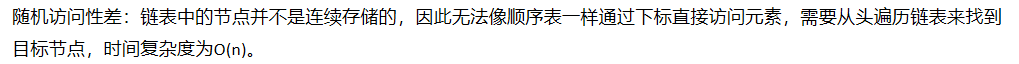
B.可随机访问任意元素

C.不必事先估计存储空间

D.所需空间与线性长度成正比

解析：

链表的特点：

* 1. 
  2. 插入和删除高效：由于链表的节点可以灵活地插入和删除，不需要移动大量元素。只需修改指针的指向，时间复杂度为O(1)。
  3. 
  4. 



在等概率情况下，顺序表的插入操作要移动\_\_\_\_\_\_结点。B

A.全部

B.一半

C.三分之一

D.四分之一

解析：

设顺序表长度为n，等概率即每个元素概率一样，插入的空有n-1个，每个为1/（n-1）。

从第1个元素到第n-1个元素插入操作要移动n,n-1,n-2......2,1。

则在等概率情况下，顺序表的插入操作要移动（n+n-1+n-2+......+2+1）\*1/（n-1）=n/2个结点。

所以正确答案为B。

所以正确答案为B。

墨迹绘图
对于双向循环链表，每个结点有两个指针域next和prior，分别指向前驱和后继。在p指针所指向的结点之后插入s指针所指结点的￼操作应为？D
A.p->next = s; p->next ->prior = s;s ->prior = p; s->next = P->next;

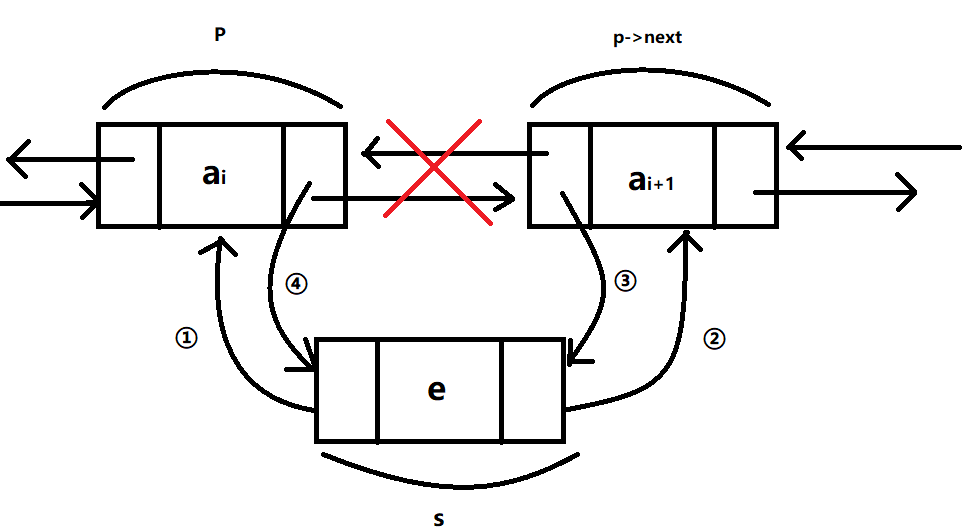

A.p->next = s; p->next ->prior = s;s ->prior = p; s->next = P->next;

B.s->prior = p; s->next = p ->next ; p ->next = s; p->next ->prior = s;

C.p ->next = s;s ->prior = p; p->next ->prior =s; s ->next = p ->next;

D.s->prior = p; s->next =p->next; p->next ->prior = s; p ->next = s;

解析：我们现在假设存储元素e的结点为s，要实现将结点s插入到结点p和p >next之间需要下面几步，如图所示。

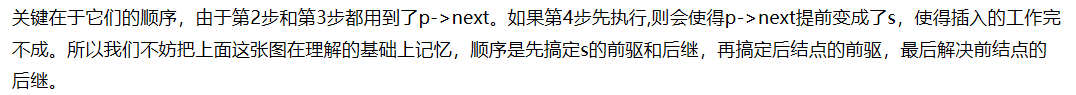


s->prior=p; /\*把p赋值给s的前驱，如图中①\*/

s->next p->next; /\*把p->next赋值给s的后继，如图中②+/

p ->next ->prior " s; /\*把s赋值给p->next的前驱，如图中③+/

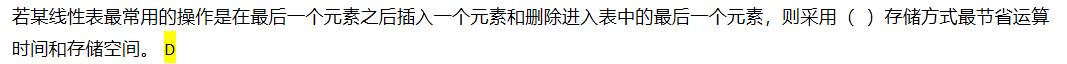
p->next - S; /\*把s赋值给p的后继，如图中④\*/



关键在于它们的顺序，由于第2步和第3步都用到了p->next。如果第4步先执行,则会使得p->next提前变成了s，使得插入的工作完￼不成。所以我们不妨把上面这张图在理解的基础上记忆，顺序是先搞定s的前驱和后继，再搞定后结点的前驱，最后解决前结点的￼后继。

墨迹绘图

若某线性表最常用的操作是在最后一个元素之后插入一个元素和删除进入表中的最后一个元素，则采用（  ）存储方式最节省运算￼时间和存储空间。 D

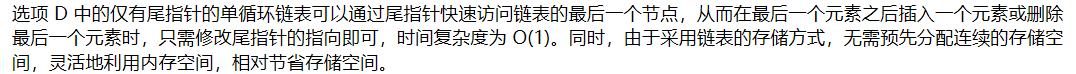
A.单链表

B.仅有头指针的单循环链表

C.双向链表

D.仅有尾指针的单循环链表







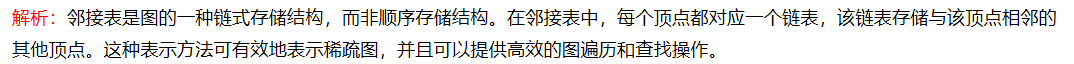

因此，选项 D 仅有尾指针的单循环链表是最适合在最后一个元素之后插入一个元素和删除最后一个元素的存储方式。
墨迹绘图

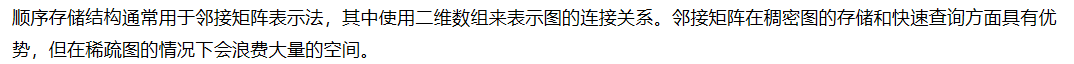
邻接表是图的一种顺序存储结构，这种说法（）B


邻接表是图的一种顺序存储结构，这种说法（）B

A.正确

B.错误





顺序存储结构通常用于邻接矩阵表示法，其中使用二维数组来表示图的连接关系。邻接矩阵在稠密图的存储和快速查询方面具有优￼势，但在稀疏图的情况下会浪费大量的空间。
因此，邻接表是一种常见的图的链式存储结构，而不是顺序存储结构。
墨迹绘图

对稀疏矩阵进行压缩存储,常用的两种方法是(    )。B

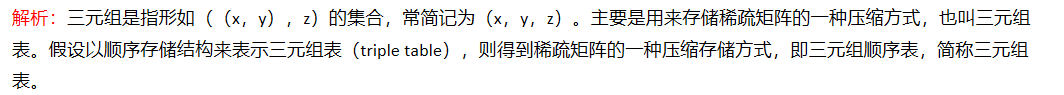

对稀疏矩阵进行压缩存储,常用的两种方法是(    )。B

A.三元组和散列表

B.三元组和十字链表

C.三角矩阵和对角矩阵

D.对角矩阵和十字链表



十字链表（Orthogonal List)是有向图的另一种链式存储结构。用十字链表来存储有向图，可以达到高效的存取效果。

散列表（Hash table，也叫哈希表），是根据关键码值(Key value)而直接进行访问的数据结构。

对角矩阵(diagonal matrix)是一个主对角线之外的元素皆为0的矩阵，常写为diag（a1，a2,...,an) 。

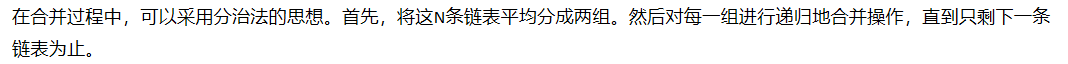
三角矩阵是方形矩阵的一种，因其非零系数的排列呈三角形状而得名。三角矩阵分上三角矩阵和下三角矩阵两种。

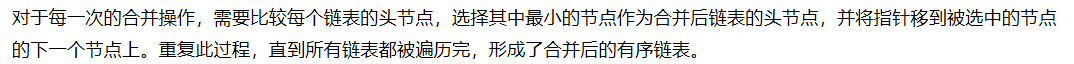
三角矩阵是方形矩阵的一种，因其非零系数的排列呈三角形状而得名。三角矩阵分上三角矩阵和下三角矩阵两种。 
稀疏矩阵进行压缩存储,常用的两种方法时三元组和十字链表，正确答案为B。
墨迹绘图
将N条长度均为M的有序链表进行合并，合并以后的链表也保持有序，时间复杂度为（） A
A.O(N * M * logN)               B.O(N*M)                C.O(N)                D.O(M)


A.O(N \* M \* logN) B.O(N\*M) C.O(N) D.O(M)

解析：

将N条长度均为M的有序链表进行合并，合并后的链表也保持有序。假设将这N条链表分别编号为1至N。





时间复杂度分析如下：

- 在每一次的合并操作中，需要遍历N个链表的头节点，找出最小节点并将指针移动到下一个节点。这个操作的时间复杂度为O(N)。

- 总共需要进行logN次合并操作，因为每次都将N条链表平均分成两组，直到只剩下一条链表。

- 每次合并操作的链表节点数量为M，所以每次合并操作的时间复杂度为O(M)。

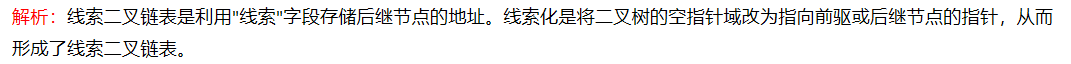
综上所述，时间复杂度为O(N \* M \* logN)。

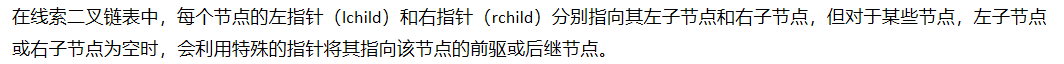
综上所述，时间复杂度为O(N * M * logN)。
因此，选项A "O(N * M * logN)" 是正确的答案。
墨迹绘图

线索二叉链表是利用（）域存储后继结点的地址。C


线索二叉链表是利用（）域存储后继结点的地址。C

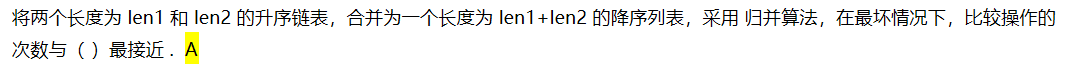
A.lchild B.data C.rchild D.root





在线索二叉链表中，每个节点的左指针（lchild）和右指针（rchild）分别指向其左子节点和右子节点，但对于某些节点，左子节点￼或右子节点为空时，会利用特殊的指针将其指向该节点的前驱或后继节点。
具体地说，可以利用ltag和rtag两个标志位来表示左指针和右指针的类型。当lchild指针指向左子节点时，ltag标志位为0；当lchild指￼针指向该节点的前驱节点时，ltag标志位为1。同样地，当rchild指针指向右子节点时，rtag标志位为0；当rchild指针指向该节点的后￼继节点时，rtag标志位为1。
墨迹绘图

将两个长度为 len1 和 len2 的升序链表，合并为一个长度为 len1+len2 的降序列表，采用 归并算法，在最坏情况下，比较操作的￼次数与（ ）最接近 .  A

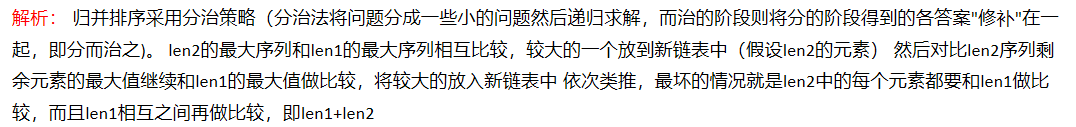



A.len1+len2

B.len1\*len2

C.min(len1,len2)

D.max(len1,len2)



**（最优情况）当两个链表一个先走完，另一个没有动的时候结果最优，最优的时间为min(len1,len2);**

（最优情况）当两个链表一个先走完，另一个没有动的时候结果最优，最优的时间为min(len1,len2);
（最坏情况）当两个链表交替进行的时候，连接两个链表的时间最长len1+len2
墨迹绘图

双循环链表中，任意一结点的后继指针均指向其逻辑后继。B


双循环链表中，任意一结点的后继指针均指向其逻辑后继。B

A.T

B.F



解析：逻辑后继:指在存储时按照需要给定的逻辑顺序在其后的数据块。循环链表中尾节点的逻辑后继应该为null而其后继指针指向￼了头节点

墨迹绘图
在一个长度为 n （ n>1 ）的单链表上，设有头和尾两个指针，执行（）操作与链表的长度有关。B
A.删除单链表中的第一个元素


A.删除单链表中的第一个元素

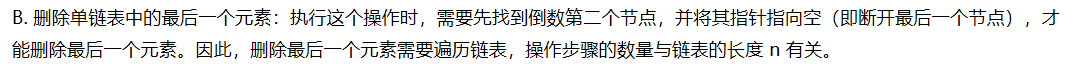
B.删除单链表中的最后一个元素

C.在单链表第一个元素前插入一个新元素

D.在单链表最后一个元素后插入一个新元素

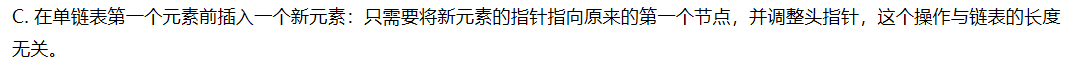
解析：

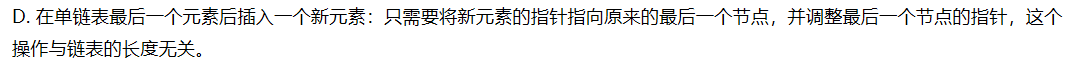
在一个长度为 n （ n>1 ）的单链表上，执行以下操作与链表的长度有关：



其他选项的操作不受链表的长度影响：

A. 删除单链表中的第一个元素：只需要将头指针指向第二个节点，这个操作与链表的长度无关。





D. 在单链表最后一个元素后插入一个新元素：只需要将新元素的指针指向原来的最后一个节点，并调整最后一个节点的指针，这个￼操作与链表的长度无关。
因此，只有删除单链表中的最后一个元素这个操作与链表的长度有关，其他操作与链表的长度无关。
墨迹绘图

已知不带头结点的双向循环链表中的结点结构为(data,last,next) ,在指针p所指结点之后插入由指针s指向的新结点，相应操作为( )?       ￼D




A.p-> next=s; s-> last=p; p->next-> last=s; S->next=p-＞next

B.p=>next=s; p-> next->last=s; S-> last= p;s-> next=p-＞next

C.s->last=p; S-> next=p->next; p-> next=s; p->next->last=s

D.s->last=p; s-> next=p-＞next; p-> next-> last=s; p-＞next=s

解析：

在指针p所指结点之后插入由指针s指向的新结点，相应操作应为选项 D：

D. s->last=p; s->next=p->next; p->next->last=s; p->next=s

解释如下：

- 首先，将新节点s的上一个指针last指向p，表示s的前一个节点是p。

- 然后，将新节点s的下一个指针next指向p的下一个节点，表示s的后一个节点是p的后一个节点。

- 接着，将p的下一个节点（原本p指向的节点）的上一个指针last指向s，表示p后面的节点的前一个节点是s。

- 最后，将p的下一个指针next指向s，表示p的后一个节点是s。


这样，就完成了在p所指结点之后插入由指针s指向的新结点的操作。注意，这是一个双向循环链表，所以需要修改四个指针来保持￼链表的循环性和链接正确。选项D中的操作正是满足这个要求的。
墨迹绘图

判断下列说法是否正确： 假设队列以不带头结点的单循环链表Q表示，只设一个指针Q->rear指向队尾元素结点(注意不设头指针)，￼且队列中元素个数大于1,  则出队时的操作是Q->rear->next = Q->rear->next->next.      A




A.正确

B.错误

data 
next 
data 
"-Sk%k, 
next 
data 
next 
next 



用数组r存储静态链表,结点的next域指向后继,工作指针j指向链中结点,使j沿链移动的操作为()？A

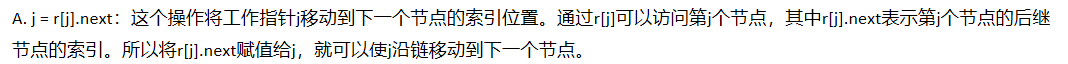
A.j=r[j].next

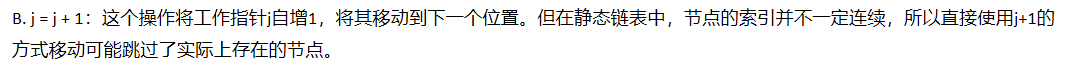
B.j=j+1

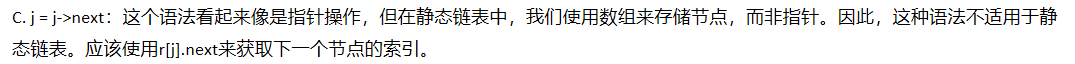
C.j=j->next

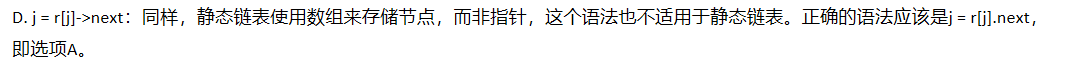
D.j=r[j]->next

解析：用数组r存储静态链表，结点的next域指向后继，并且工作指针j指向链中的结点。











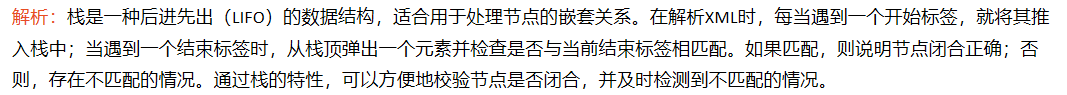
解析XML时，需要校验节点是否闭合，如必须有与之对应，用（）数据结构实现比较好 D

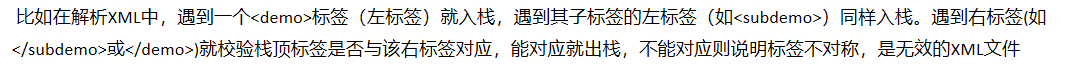
A.链表

B.树

C.队列

D.栈





因此，栈是一种较为常用的数据结构，用于校验XML节点的闭合情况。选项D即栈是实现该功能的比较好的数据结构。

因此，栈是一种较为常用的数据结构，用于校验XML节点的闭合情况。选项D即栈是实现该功能的比较好的数据结构。

墨迹绘图

若某线性表最常用的操作是存取任一指定序号的元素和在最后进行插入和删除运算,则利用（）存储方式最节省时间。A


若某线性表最常用的操作是存取任一指定序号的元素和在最后进行插入和删除运算,则利用（）存储方式最节省时间。A

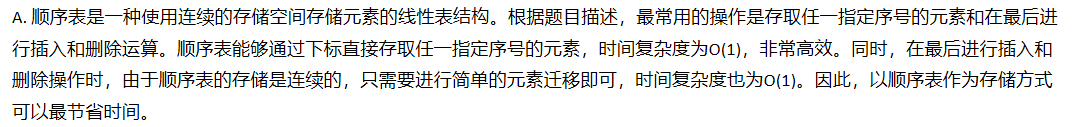
A.顺序表

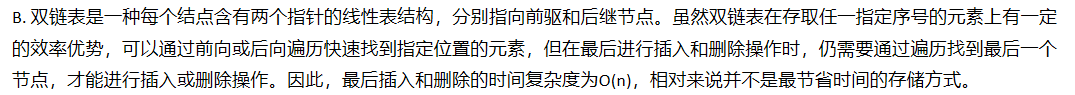
B.双链表

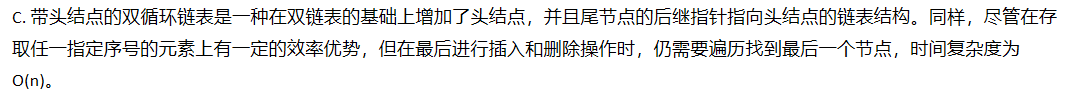
C.带头结点的双循环链表

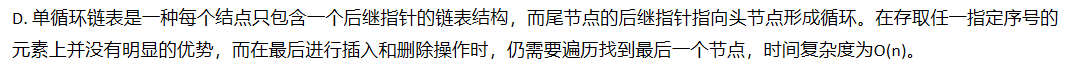
D.单循环链表

解析：











综上所述，选项A即顺序表是在这种情况下最节省时间的存储方式。

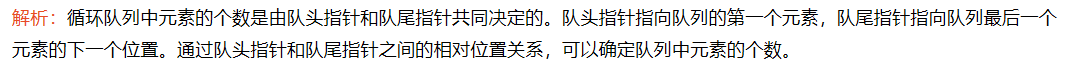
下列叙述中正确的是？D

A.循环队列有队头和队尾两个指针，因此，循环队列是非线性结构

B.在循环队列中，只需要队头指针就能反映队列中元素的动态变化情况

C.在循环队列中，只需要队尾指针就能反映队列中元素的动态变化情况

D.循环队列中元素的个数是有队头指针和队尾指针共同决定



另外选项的解释如下：

A. 循环队列的队头和队尾指针确实是循环队列的组成部分，但循环队列仍然属于线性结构，因为它的元素在内存中是连续存储的。

B. 循环队列的队头指针无法完全反映队列中的动态变化情况，还需要队尾指针来确定队列是否为空或满。

C. 同样地，循环队列的队尾指针也无法完全反映队列中的动态变化情况，还需要队头指针来确定队列是否为空或满。

因此，正确的叙述是D。

因此，正确的叙述是D。

墨迹绘图
单链表的按升序排列，其中L指向头结点，写一个函数,若start<元素<end，则删除，并释放空间。 
如13->19->26->30->49->63,start=21,end=51,删除后为13->19->63.结点定义及函数模板如下，请为（1)(2)(3)(4)处出选择合适的代码。D 


如13->19->26->30->49->63,start=21,end=51,删除后为13->19->63.结点定义及函数模板如下，请为（1)(2)(3)(4)处出选择合适的代码。D

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | typedef struct Node{      int data;      struct Node \*next;  }Node,\*LinkList;  int delList(LinkList L,int start,int end)  {      (1) \* q = L，\*p = L->next;       int delFlag = 0;       while ( p )           if ( p->data <= start)               { q = p; p = (2);}           else               if ( p->data < end )               {                   q->next = (3); free(p);                   p = (4); delFlag = 1;                }               else  break;        if( delFlag == 0) return -1;        return 0;  } |

A.(1)LinkList (2)p->next (3)p (4)q->next

B.(1)LinkList (2)p->next (3)p->next (4)q->next

C.(1)Node (2)p->next (3)p->next (4)q

D.(1)Node (2)p->next (3)p->next (4)q->next

解析：

（1）已经定义了指针，所以用Node。

（2）移动节点无需删除，p = p->next。

（3）需要删除的节点，q->next = p->next。

（4）操作完成后需要移动指针p，p=q->next，易错为p=p->next。

正确答案为D。

NumberList是一个顺序容器，以下代码执行后，NumberList里的元素依次为： C

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | List<int> NumberList = new List<int>(){2,4,1,3,5};  for(int i = 0;i<NumberList.Count;++i)  {      int v = NumberList[i];      if(v%2 = = 0)      {          NumberList.Remove(v);//删除的是元素，而非下标      }  } |

A.2,4,1,3,5

B.2,1,3,5

C.4,1,3,5

D.1,3,5

D.1,3,5

墨迹绘图
删除非空线性链表中由指针p所指链结点的直接后继结点的过程是依次执行动 
作（）。(设链结点的构造为） B


作（）。(设链结点的构造为） B



A.r←link(p); link(p)←r; call RET(r)

B.r←link(p); link(p)←link(r); call RET(r)

C.r←link(p); link(p)←link(r); call RET(p)

D.link(p)←link(link(p)); call RET(p)

解析：

节点p,要删除p->next;

r = p->next; p->next = r->next; depose(r)



若频繁删除某线性表的第1个元素，则不宜采用以下哪种存储方式（）B

A.单链表

B.顺序表

C.单向循环链表

D.双链表


解析：不宜采用顺序表（B）存储方式，因为顺序表的元素在内存中是连续存储的，删除第一个元素需要将后面的元素向前移动，￼导致效率较低。 而其他三种存储方式（单链表、单向循环链表、双链表）可以通过修改指针来快速删除第一个元素，效率较高。 
墨迹绘图

将长度为n的单链表链接在长度为m的单链表后面，其算法的时间复杂度釆用大O形式表示应该是（ ）C


将长度为n的单链表链接在长度为m的单链表后面，其算法的时间复杂度釆用大O形式表示应该是（ ）C

A.O(1)

B.O(n)

C.O(m)

D.O(n+m)

解析：需要先遍历长度为m的单链表，找到链表尾部，然后用尾部指针指向长度为n单链表的头结点。

所以时间复杂度只与长度为m的单链表有关，时间复杂度为O(m)，正确答案为C。

下列叙述中不正确的是（    ）ACD

A.在链表中，如果每个结点有两个指针域，则该链表一定是非线性结构

B.在链表中，如果有两个结点的同一个指针域的值相等，则该链表一定是非线性结构

C.在链表中，如果每个结点有两个指针域，则该链表一定是线性结构

D.在链表中，如果有两个结点的同一个指针域的值相等，则该链表一定是线性结构


解析：A的反例为双向链表，每个节点都有两个指针域，但是其属于线性结构。C的反例为二叉树，二叉树的每个节点都包含两个指￼针域(一个指向左孩子，一个指向右孩子)，但是其属于非线性结构。对于B,D的话，如果链表里存在两个节点指向同一个指针的话，￼那么它注定是非线性结构(线性结构要求只有一个头和一个尾)。
墨迹绘图

typedef struct node_s{


typedef struct node\_s{

int item;

struct node\_s\* next;

}node\_t;

node\_t\* reverse\_list(node\_t\* head)

{

node\_t\* n=head;

head=NULL;

while(n){

\_\_\_\_\_\_\_\_\_

}

return head;

}

空白处填入以下哪项能实现该函数的功能？B

A.node\_t\* m=head; head=n; head->next=m; n=n->next;

B.node\_t\* m=n; n=n->next; m->next=head; head=m;

C.node\_t\* m=n->next; n->next=head; n=m; head=n;

D.head=n->next; head->next=n; n=n->next;

解析：

* 1. 声明了一个局部变量 n，初始化为传入的头节点。
  2. 将传入的头节点 head赋值为 NULL，将反转后的链表的头节点指针先设为空。
  3. 进入循环，每次迭代都检查当前节点 n是否为 NULL，如果是则跳出循环。
  4. 在循环中，首先在内部定义了一个指针变量 m，将其指向当前节点 n。
  5. 然后将当前节点 n的下一个节点保存在 n中（即向后移动一个节点）。
  6. 将 m的 next指针指向之前链表反转的部分，即指向原始链表中的前一个节点。
  7. 将 head更新为 m，使其成为反转链表的新头节点。
  8. 回到循环开始，继续处理下一个节点，直到遍历完整个链表。
  9. 返回反转后的链表的头节点指针 head。



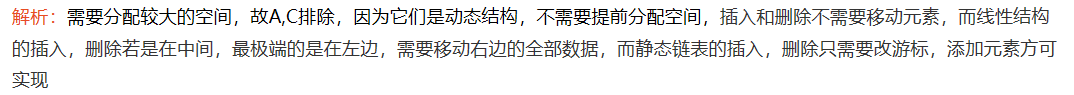
需要分配较大空间，插入和删除不需要移动元素的线性表，其存储结构是() 。B

A.单链表

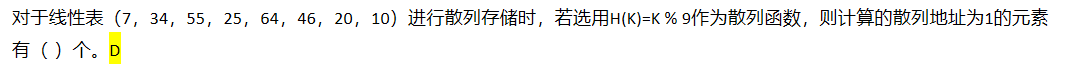
B.静态链表

C.线性链表

D.顺序存储结构







A.1

B.2

C.3

D.4

解析：

若选用散列函数 H(K) = K % 9，对线性表 (7, 34, 55, 25, 64, 46, 20, 10) 进行散列存储，计算散列地址为1的元素个数如下：

* 1. 对于每个元素 K，计算 H(K) = K % 9 的值。
  2. 若计算结果与所需的散列地址相匹配（即等于1），则计数器加1。



设链式栈中结点的结构为（data ,link)，且top是指向栈顶的指针，若想在链式栈的栈顶插入一个由指针s所指的结点，则应执行（  ）操作。C

A.top->link=s;

B.s->link=top->link; top->link=s;

C.s->link=top; top=s;

D.s->link=top; top=top->link;

解析：

* 1. 首先，将新节点 s 的 link 指向当前栈顶元素，即原来栈顶元素的位置，表示新节点将成为新的栈顶。
  2. 然后，将 top 指针指向新插入的结点 s，使其成为新栈顶。



以下关于链表和数组说法正确的是（）BC

A.数组从栈中分配空间，链表从堆中分配空间

B.数组插入或删除元素的时间复杂度O(n)，链表的时间复杂度O(1)

C.数组利用下标定位，时间复杂度为O(1)，链表定位元素时间复杂度O(n）

D.对于add和remove，ArrayList要比LinkedList快

解析：

A. 数组和链表都可以从栈或堆中分配空间，具体取决于它们的创建方式和作用域，并不能一概而论。

B. 数组插入或删除元素的时间复杂度通常为O(n)，因为需要移动其他元素来保持连续性，而链表的插入和删除操作通常是在O(1)的时间复杂度内完成，因为只需要调整指针的指向即可。

D. 对于 add 和 remove 操作，在某些情况下 ArrayList 可能比 LinkedList 快，但这取决于具体的操作和数据规模。在插入和删除元素方面，LinkedList 在某些情况下可能更快，因为不需要移动其他元素。因此，它们的性能会受到具体使用情况和操作类型的影响。

C. 数组利用下标定位，时间复杂度为O(1)，链表定位元素时间复杂度O(n）。

解释如下：

* 1. 在数组中，通过下标索引可以直接访问特定位置的元素，因此定位元素的时间复杂度为常数时间O(1)。
  2. 而在链表中，需要从头节点开始逐个遍历元素，直到找到目标元素，因此定位元素的时间复杂度为O(n)。需要注意的是，若链表是有序的，则可以使用一些优化技巧提高定位的效率，但大体上仍然相对较慢。



树的高度是指根到叶子节点的最长路径的边数(根的高度为0)。一个二叉树的中序遍历序列为 KAFDEBGC，前序遍历序列为 EFAKDGBC，则该二叉树的高度为 3

解析：

给定中序遍历序列 KAFDEBGC 和前序遍历序列 EFAKDGBC，我们可以按照以下步骤重建二叉树：

* 1. 根据前序遍历序列确定根节点为 E。
  2. 根据中序遍历序列将节点分成左子树和右子树的部分：
     + KAFD 为左子树中序遍历序列，对应的前序遍历序列为 FAKD。
     + GC 为右子树中序遍历序列，对应的前序遍历序列为 DGCB。

重建得到的二叉树如下所示：

E

/ \

F G

/ \ /

A K C

/ \

D B

在这个二叉树中，从根节点到叶子节点的最长路径是 3（经过节点 F、A、K 的路径）。因此，该二叉树的高度为 3。

答案：3



在有n个结点的二叉链表中，值为非空的链域的个数为（ ）。A

A.n-1

B.2n-1

C.n+1

D.2n+1

解析：值为空的链域个数为n+1个，为非空的个数为n-1



线性表的顺序存储结构是一种\_\_\_\_\_\_的存储结构。A

A.随机存取

B.顺序存取

C.索引存取

D.散列存取

解析：

1、随机存取就是直接存取，可以通过下标直接访问到元素的位置，与存储位置无关，时间复杂度永远为O(1)，例如数组。存取第N个数据时，不需要访问前（N-1）个数据，直接就可以对第N个数据操作 （array）。

2、非随机存取也叫顺序存取，不能通过下标访问，在存取第N个数据时，必须先访问前（N-1）个数据 ，例如链表。



从一个具有 n 个结点的单链表中查找其值等于 x 结点时，在查找成功的情况下，需平均比较（ ）个结点。D

A.n

B.n/2

C.(n-1)/2

D.(n+1)/2

解析：

在一个具有 n 个结点的单链表中查找值等于 x 的结点时，平均比较的结点个数可以看作是成功查找所需要进行的平均比较次数。

由于单链表只能从头节点开始顺序查找，每次比较都需要遍历至多 n 个结点。 在最坏情况下，需要遍历到链表的最后一个结点才能找到目标结点。

因此，平均比较的结点个数为 (n+1)/2。



线性表的顺序存储方式具有哪些优点（）ABC

A.可以用结点的物理次序反映结点之间的逻辑关系

B.存储密度高，节省存储空间

C.在结点等长时可以随机存取

D.插入和删除比较灵活

解析：

顺序存储：在一块连续的存储区域一个接着一个的存放数据。顺序存储方式把逻辑上相邻的节点存储在物理位置放在相邻的存储单元里，节点间的逻辑关系由存储单元的邻接关系来体现。顺序存储方式也称为顺序存储结构，一般采用数组或结构数组来描述。 优点： 在结点等长时可以随机存取 存储密度高节省存储空间 用结点的物理次序反映结点之间的逻辑关系 缺点： 插入和删除结点时要移动大量的结点 必须静态分配连续空间

链接存储：链接存储方式比较灵活，不要求逻辑上相邻的节点在物理位置上相邻，节点间的逻辑关系由附加的引用字段来表示。一个节点的引用字段往往指向下一个节点的存放位置。链接存储方式也成为链式存储结构。 优点： 插入和删除比较灵活，不需要大量移动结点 动态分配空间比较灵活，不需要预先申请最大的连续空间 缺点： 增加指针的空间开销 检索必须沿链进行，不能随机存取 因此D不是顺序存储的优点，而是链接存储的优点。



将长度为n的单链表链接在长度为m的单链表之后的算法的时间复杂度为(  )？C

A.0(1)

B.0(n)

C.0(m)

D.0(m+n)

解析：

考察的是合并两个单链表的时间复杂度。 根据题目的描述属于“尾插法”。 将长度为m的单链表头部固定，设立一个指针进行向尾部搜索，找到尾部的时间复杂度为O(m) 搜索长度为n单链表的头结点，时间复杂度为O(1) 所以总的时间复杂度为O(m)



Which statement is true for the class java.util.ArrayList? A

下面那个选项有关java.util.ArrayList是正确的

A.The elements in the collection are ordered.

集合中的元素是排序的

B.The collection is guaranteed to be immutable.

集合不可改变

C.The elements in the collection are guaranteed to be unique.

集合中的元素必须唯一

D.The elements in the collection are accessed using a unique key.

集合中元素的键是唯一的

E.The elements in the collections are guaranteed to be synchronized.

集合中的元素是线程同步的

解析：

在一个 ArrayList，元素按添加顺序存储，并可通过其索引位置进行访问。这意味着元素的顺序保持不变，允许可预测的迭代和检索。

其他陈述不适用于 ArrayList：

* 1. B.该集合保证是不可变的： ArrayList 允许修改其元素。它是一个动态数组，可以通过添加、删除或修改元素来修改。
  2. C.集合中的元素保证是唯一的： ArrayList 允许重复元素。
  3. D.集合中的元素使用唯一键访问： ArrayList 使用索引访问元素，而不是唯一键。
  4. E.保证集合中的元素同步： ArrayList 未同步。如果需要线程安全，则需要显式实现同步机制。



单链表结点的数据元素只能是哪一种？（  ） C

A.整型

B.字符串

C.任何数据类型

D.实型

解析：

单链表是一种链式存取的数据结构，用一组地址任意的存储单元存放线性表中的数据元素。每个结点的构成：元素（数据元素的映象) + 指针（指示后继元素存储位置) 其数据元素没有特定的类型限制。



关于双链表的搜索给定元素操作的说法正确的是? B

A.从两个方向搜索双链表，比从一个方向搜索双链表的速度慢

B.从两个方向搜索双链表，比从一个方向搜索双链表的方差要小

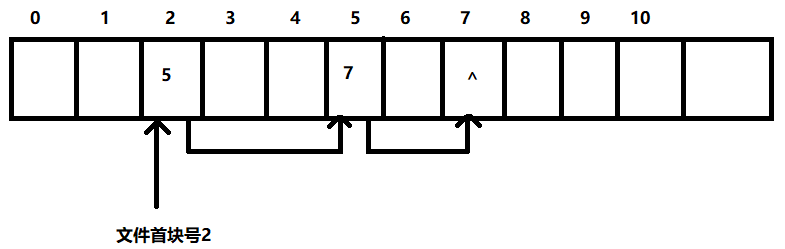
C.从两个方向搜索双链表，比从一个方向搜索双链表速度要快

D.以上说法都不正确

解析：如果链表数据是无序的，则单向搜索与双向搜索平均速度相同 如果链表是有序的，而要搜索的数据距离最小值（最大值）较近，这种情况下双向搜索平均速度更快。 因此双向搜索更稳定，方差更小



文件分配表FAT是管理磁盘空间的一种数据结构，用在以链接方式存储文件的系统中记录磁盘分配和跟踪空白磁盘块。整个磁盘仅设一张FAT表，其结构如下图所示。如果文件块号为2，查找FAT序号为2的内容得知物理块2的后继物理块是5；再查FAT序号为5的内容得知物理块5的后继物理块是7；接着继续查FAT序号为7的内容为“^”，即该文件结束标志，所以该文件由物理块2、5、7组成。



假设磁盘物理块大小为1KB，并且FAT序号以4bits为单位向上扩充空间。请计算下列两块磁盘的FAT最少需要占用多大的存储空间？

1）一块540MB的硬盘

2）一块1.2GB的硬盘 C

A.1M 3M

B.1.2M 3.15M

C.1.35M 3.6M

D.1.3M 3.15M

解析：

（1）由题设条件可知，硬盘大小为540MB且磁盘物理块大小为1KB时，该硬盘共有盘块540MB/1KB = 540K个。 又因为219<540K<220，所以540K个盘块号需要用20位二进制数表示，也即文件分配表FAT的每个表项为20/8＝2.5B。所以，540MB磁盘的FAT需占用存储空间容量为： 2.5B×540K＝1350KB （2）当硬盘容量大小为1.2GB时，硬盘共有盘块1.2M个。 又因为220<1.2M <221，所以1.2M个盘块号需要用21位二进制数表示，为了方便FAT的存取，则每个表项用24位（3B）二进制数表示。所以，1.2GB磁盘的FAT需占用存储空间容量为： 3B×1.2M＝3.6MB

1B = 8bit

序号只能是4的倍数。

物理块大小1KB，540MB，即540K个分区，计算 2^19 < 540K < 2^20，所以序号是20个bit，且20是4的倍数。每个序号需要20 / 8 = 2.5B的空间存储。

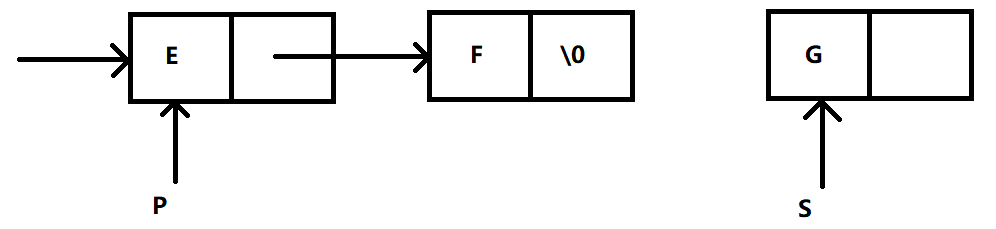
540MB需要540K \* 2.5B = 1350KB即1.35M 。

1.2GB，即1200K个分区，2^20 < 1200K < 2^21，也就是需要21个bit才能存储，但是序号只能是4个倍数，所以需要24个bit，也就是3B的空间。

1.2GB需要1200K \* 3B = 3600KB即3.6M



 若建立以下链表结构,指针P,S分别指向如图所示结点



则不能将S所指结点插入到链表末尾的语句组是? A

A.p=p-> next; s->next=p;p-> next=s;

B.s->next= '\0'; p=p-> next; p-> next=s;

C.p=p-> next; s ->next=p-> next;p-> next=s;

D.p= ( \*p ).next; ( \* s ).next=( \* p ).next; ( \*p ).next=s;

解析：

1> 对应B选项

     把s结点的next指向[null](https://www.baidu.com/s?wd=null&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9mWnYmWfLuH--rjPhmvnv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHbYPW01PjR4rHc1PHndnjRYr0)即：s->next = '\0';

     把p指针指向F所在的结点，也就是p结点的next指针所指向的结点：p = p->next;

     把F所在的结点的next指向s结点；

2>对应C选项

     p指针指向F所在的结点：p = p->next;

     s结点的next指针指向p->next,也就是[NULL](https://www.baidu.com/s?wd=NULL&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1d9mWnYmWfLuH--rjPhmvnv0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHbYPW01PjR4rHc1PHndnjRYr0)：s->next = p->next;

     把F所在的结点的next指向s结点；

3>对应选项D

     这个选项其实和C选项采用的方法是一样的，只不过取值的方式不一样，C选项用指针取值的,而D选项首先把指针的所指向的地址给拿出来，在对其取值，相当于普通变量。所以C选项用的是'->'而D选项用的是'.'



一个长度为100的循环链表，指针A和指针B都指向了链表中的同一个节点，A以步长为1向前移动，B以步长为3向前移动，一共需要同时移动多少步A和B才能再次指向同一个节点\_\_\_\_。E

A.99

B.100

C.101

D.49

E.50

F.51

解析：

这道题不太行

纯属数学问题，设经过x步二者重合，3x - x = 100\*n  即 x = 50\*n，x为50的整数倍，最小为50  
其他类的问题也可以通过数学表达式解决（n1 - n2）\*x = lenth\*n   
n1:移动快的步数  
n2:移动慢的步数  
n：经过n圈（二者重合，快的指针比慢的指针，多走n圈）  
lenth：链表的长度



某一系统功能，需要一次性加载N（N在1000左右）个随机数，后续只对该集合进行遍历。最宜采用哪种结构存放？ C

A Hash表 B 二叉树 C 链表 D 图

解析：

链表是一种简单的数据结构，它可以快速插入和删除元素，不需要像数组那样进行扩容和复制操作。在这个特定的功能需求下，我们只需要一次性加载N个随机数，并后续只进行遍历操作，而不需要频繁地进行插入、删除和查找等操作。

使用链表存放N个随机数的好处有两个：

1. 空间效率高：链表可以根据需要动态分配内存，不会浪费额外的空间。

2. 时间效率高：由于后续只进行遍历操作，链表的遍历时间复杂度是O(N)，与N的大小成线性关系，非常高效。

相比之下，选项 A（Hash表）虽然可以提供常数时间的插入、删除和查找操作，但在仅进行遍历的场景下，使用Hash表会浪费额外的空间和引入额外的复杂性。

选项 B（二叉树）和选项 D（图）都不适合这个场景，因为它们的主要优势是支持高效的查找操作，而在本题中并不需要频繁的查找操作。

综上所述，最宜采用的数据结构是链表（选项 C）来存放这些随机数。



广义表即我们通常所说的列表（lists)。它放松了对表元素的原子性限制，允许他们有自身结构。那么广义表E((a,(a,b),((a,b),c)))的长度和深度分别为：B

A.2和4

B.1和4

C.1和3

D.2和3

解析：

广义表的长度： 若广义表不空，则广义表所包含的元素的个数，叫广义表的长度，数第一层括号内的逗号数目。

广义表的深度： 广义表中括号的最大层数叫广义表的深度。

(a,(a,b),((a,b),c)) 
a (a,b) ((a,b),c) 



线性表的长度是线性表所占用的存储空间的大小。 B

A.T

B.F

解析：

线性表的长度是指线性表中元素的个数，并不直接与线性表所占用的存储空间的大小有关。线性表的存储空间大小可能受到多种因素的影响，例如底层数据结构的选择、元素类型的大小等等。



下列哪些容器可以使用数组，但不能使用链表来实现? D

A.队列

B.栈

C.优先级队列

D.Map或者Dict

解析：

Map或Dict是一种键值对集合，可以用于存储和检索数据。它可以使用数组来实现，称为哈希表（Hash Table）或散列表。哈希表通过将键映射到数组的索引位置来实现高效的插入、删除和查找操作。具体而言，哈希表使用哈希函数将键转换为数组的索引，然后在该索引位置存储对应的值。当存在多个键映射到同一个索引位置时，可以使用链表或其他解决冲突的方法来处理。

相反，链表不适合直接实现Map或Dict，因为链表的查找操作需要遍历整个链表，时间复杂度为O(n)，不如哈希表的O(1)高效。



若广义表A满足Head(A) = Tail (A), 则A为？B

A.( )

B.( ( ) )

C.( ( ), ( ) )

D.（ ），（ ），（ ））

解析：

广义表第一个元素是表头，其余元素是表尾，如果只有一个元素，那么表尾为空即（），

B中head(A)=();tail(A)=();

但是在选项C中，head(A)=();tail(A)=（（））;

D中head(A)=);tail(A)=((),());



如果使用比较高效的算法判断单链表有没有环的算法中，至少需要几个指针？B

A.1

B.2

C.3

D.4

解析：两个。 一个每次走一步， 一次每次走两步， 会相遇就表示有环



下列叙述哪些是对的? CD

A.线性表的逻辑顺序与物理顺序总是一致的。

B.线性表的顺序存储表示优于链式存储表示。

C.线性表若采用链式存储表示时所有结点之间的存储单元地址可连续可不连续。

D.二维数组是每个元素都为线性表的线性表 .

E每种数据结构都应具备三种基本运算：插入、删除和搜索。

解析：

A中线性表的顺序存储结构中逻辑顺序与物理顺序总是一致的，但链式存储结构却不是，

B：两种存储结构没有优劣之分，只是看在什么情况下适合哪种存储结构。对于经常插入或删除的

适合选择链式存储，经常查找的适合顺序存储。

C

D正确二维数组是一种特殊的数组，它可以看作是由多个一维数组组成的数据结构。将每个一维数组看作是线性表，这里题目的意思就是二维数组的元素是一维数组组成的

E中数据结构的基本运算中，搜索肯定不是必须的，比如栈中就没有搜索



用邻接表表示图进行深度优先遍历时,通常是采用（）来实现算法的。 A

A.栈

B.队列

C.树

D.图

解析：

深度DFS：需要递归，使用顺序栈；

深度优先遍历（Depth First Search，DFS）是一种图遍历的方法，其基本思想是从起始节点开始，探索尽可能深的路径，直到无法继续前进时回溯到上一个节点，然后选择下一个未探索的节点继续探索。在实现过程中，可以使用栈来存储待访问节点，每次从栈顶取出一个节点进行探索，并将其未访问的邻居节点入栈。

广度BFS：类似层次遍历；需要循环队列

广度优先遍历（Breadth First Search，BFS）是一种图遍历的方法，其基本思想是从起始节点开始，先访问离起始节点最近的节点，然后逐层向外扩展，直到遍历完所有可达节点。在实现过程中，可以使用队列来存储待访问节点，每次从队列头取出一个节点进行访问，并将其邻居节点入队。



某表达式的前缀形式为"+-\*^ABCD/E/F+GH",它的中缀形式为() C

A.A^B\*C-D+E/F/G+H

B.A^B\*(C-D)+(E/F)/G+H

C.A^B\*C-D+E/(F/(G+H))

D.A^B\*(C-D)+E/(F/(G+H))

解析：

前缀表达式的计算机求值特点：

从右至左扫描表达式，遇到数字时，将数字压入堆栈，遇到运算符时，弹出栈顶的两个数，用运算符对它们做相应的计算（栈顶元素 op 次顶元素），并将结果入栈；重复上述过程直到表达式最左端，最后运算得出的值即为表达式的结果。

根据从右至左扫描计算过程如下:

题目中的前缀形式为：+-\*^ABCD/E/F+GH

1) 首先扫描 H,是数字 入栈 ，栈中为: H

2) 扫描 G 为数字 入栈 ,栈中为:G,H

3)扫描+ 为运算符 ,依次弹出G ,H ,得到 G+H 的结果 入栈，栈中为： G+H(在这里方便讲解 标记为  G+H)

4)扫描 F 为数字 ,入栈 ，栈中为  F, G+H

5)扫描 / 为运算符, 依次弹出 F,G+H ，计算F/(G+H) 的结果入栈 ，栈中为 F/(G+H)

6)扫描 E 为数字,入栈，栈中为 E, F/(G+H)

7)扫描 / 为运算符, 依次弹出E, F/(G+H) ，计算 E/(F/(G+H))

8)扫描 D 为数字,入栈 栈中为:D,E/(F/(G+H))

9)扫描 C 为数字,入栈 栈中为:C,D,E/(F/(G+H))

10)扫描 B 为数字,入栈 栈中为:B,C,D,E/(F/(G+H))

11)扫描 A 为数字,入栈 栈中为:A,B,C,D,E/(F/(G+H))

12) 扫描^ 为数字,依次弹出 A,B 计算 A^B的结果入栈, 栈中为：A^B,C,D,E/(F/(G+H))

13) 扫描\*为数字,依次弹出 A^B,C 计算 A^B\*C的结果入栈, 栈中为：A^B\*C,D,E/(F/(G+H))

14) 扫描-为数字,依次弹出 A^B\*C,D 计算 A^B\*C-D的结果入栈, 栈中为：A^B\*C-D,E/(F/(G+H))

15) 扫描+为数字,依次弹出 A^B\*C-D,E/(F/(G+H))  计算 A^B\*C-D+ E/(F/(G+H)) 的到结果

最后得到的表达式为： A^B\*C-D+E/(F/(G+H))



已知操作符包括‘ +’、‘ -’、‘ \*’、‘ /’、‘ (’和‘ )’。将中缀表达式 a+b-a\*((c+d)/e-f)+g 转换为后缀表达式 ab+acd+e/f-\*-g+时，用栈来存放暂时还不能确定运算次序的操作符。若栈初始时为空，则转换过程中同时保存在栈中的操作符的最大个数是（ ）。 A

A.5

B.7

C.8

D.11

解析：

a,b.+.a 
a,Et, a.c. 
a.b.t, 
Lb f. a, c.d.t 
09 _ 
— , 4,1- 
a.bt,a, C, 



一个栈的输入序列为连续整数1，2，3...n，若输出序列的第一个元素是n，输出第  i（1<=i<=n）个元素是（  ）。B

A.不确定

B.n-i+1

C.I

D.N-I

解析：

按照进栈的顺序，先进后出，只有所有的元素进入栈内，n才会是第一个出来，所以第二个出来的是n-1,第i个出来的是n-i+1



下列叙述中正确的是（ ）B

A.在栈中，栈顶指针的动态变化决定栈中元素的个数

B.在循环队列中，队尾指针的动态变化决定队列的长度

C.在循环链表中，头指针和链尾指针的动态变化决定链表的长度

D.在线性链表中，头指针和链尾指针的动态变化决定链表的长度

解析：

A. 在栈中，栈顶指针的动态变化并不直接决定栈中元素的个数。栈是一种后进先出（LIFO）的数据结构，栈顶指针指向栈顶元素，其动态变化反映了栈的状态，但并不能直接用来确定栈中元素的个数。

B. 在循环队列中，队尾指针的动态变化决定队列的长度。

在循环队列中，队尾指针指向最后一个元素的下一个位置，可以通过队尾指针来确定队列的长度。当队列为空时，队尾指针与队头指针相等；当队列满时，队尾指针与队头指针之间有一个空位。通过队尾指针的动态变化，我们可以确定队列当前的长度。

C. 在循环链表中，头指针和链尾指针的动态变化并不决定链表的长度。循环链表是一种特殊的链表，它的最后一个节点指向头节点形成一个闭环。在循环链表中，头指针和链尾指针的动态变化主要用于维护循环链表的结构，并不能直接决定链表的长度。

D. 在线性链表中，头指针和链尾指针的动态变化也不决定链表的长度。在线性链表中，每个节点通过指针连接，而头指针指向链表的第一个节点，链尾指针指向链表的最后一个节点。它们的动态变化通常用于操作链表的插入、删除等操作，而不能直接决定链表的长度。

综上所述，只有选项B是正确的。



元素a，b，c，d，e依次进入初始为空的栈中，若元素进栈后可停留、可出栈，直到所有元素都出栈，则在所有可能的出栈序列中，以元素d开头的序列个数是（）。B

A.3 B.4 C.5 D.6

解析：

d为第1个出栈元素，则d之前的元素必定是进栈后在栈中停留。因而出栈顺序必为d\_c\_b\_a\_，e的顺序不定，在任一“\_”上都有可能，一共有4种可能。

【另解】d首先出栈，则abc停留在栈中，此时栈的状态如下图所示。



此时可以有如下4种操作：①e进栈后出栈，则出栈序列为decba；②c出栈，e进栈后出栈，出栈序列为dceba；③cb出栈，e进栈后出栈，出栈序列为dcbea；④cba出栈，e进栈后出栈，出栈序列为dcbae。



对中缀表达式 3\*2^(4+2\*2-6\*3)-5 进行求值，求值过程中当扫描到6时，对象栈和算符栈分别为（），其中^为乘幂。D

A.3，2，8; \*^-

B.3，2，4，2，2; \*^+\*-

C.3，2，4，2，2; \*^(+\*-

D.3，2，8; \*^(-

解析：

酽 2 ^ （ 4 + 2 · 驴 《 卜 5 
对 象 懂 
3 
3 ， 2 ， 4 2 ， 2 
运 算 栈 
2 ， 2 出 栈 ， 进 4 
注 ： 此 处 ^ 优 先 级 高 ， 所 以 并 未 合 并 
． 违 ： ． 的 优 先 级 低 合 抖 
注 ： 
+ ， ． 同 级 ， + 出 栈 



若一序列进栈顺序为e1,e2,e3,e4,e5,问存在多少种可能的出栈序列（）B

A.41

B.42

C.43

D.44

解析：

卡 特 兰 数 的 一 个 代 表 例 子 丿 0 ） ： 42 
出 栈 方 式 数 量 的 题 可 以 这 么 理 解 ： 
设 视 个 数 出 栈 方 式 数 量 为 八 
因 为 入 栈 的 顺 序 是 确 定 的 ， 下 标 记 为 1 、 23 “ 
设 最 后 一 个 出 栈 的 是 第 k 个 数 ， 则 说 明 k 之 前 的 一 1 个 数 
先 完 成 进 栈 出 栈 ， 有 八 k 一 1 ） 种 方 式 ； 然 后 k 之 后 的 一 k 个 
数 再 完 成 进 栈 出 栈 ， 有 熹 视 一 k ） 种 方 式 ： 最 后 第 k 个 数 出 栈 。 
这 时 就 有 八 k 一 1) * 八 一 k ） 种 方 式 
而 每 个 数 都 有 可 能 是 最 后 出 栈 的 
若 一 序 列 进 栈 顺 序 为 el ， e2 ， e3 ， e4 ， e5 ， 问 存 在 多 少 种 可 能 的 出 栈 序 列 
所 以 有 八 ， t) = 八 0 ） *f(n 一 1 ） + 八 1 ） * 八 n 一 2 ） + ． 
上 面 八 0 ） ： 1 是 为 了 写 成 统 一 式 ： 丿 和 1) ： 乥 丿 一 l) · 丆 一 幻 
由 上 式 的 通 项 就 是 卡 特 兰 数 丿 佃 ） = 
“ + 1 
分 支 主 題 



设输入序列是1,3,5....m,经过栈的作用后输出序列的第一个元素是m,则输出序列中第i个输出元素是() A

A.m-2(i-1)

B.m-i

C.m-1-i

D.m+1-i

解析：

第一个为M

第二个为m - 2\*(2-1)

第三个为m-2\*(3-1)

.....

第i 个为m-2\*(i-1)



如下C++程序

|  |  |
| --- | --- |
| 1  2  3 | int i=0x22222222；  char szTest[]="aaaa";  //a的ascii码为0x61  func(I, szTest);    //函数原型为void func(int a,char \*sz); |

请问刚进入func函数时，参数在栈中的形式可能为 （左侧为地址，右侧为数据—） D

A.0x0013FCF0 0x61616161 0x0013FCF4 0x22222222 0x0013FCF8 0x00000000

B.0x0013FCF0 0x22222222 0x0013FCF4 0x0013FCF8 0x0013FCF8 0x61616161

C.0x0013FCF0 0x22222222 0x0013FCF4 0x61616161 0x0013FCF8 0x00000000

D.0x0013FCF0 0x0013FCF8 0x0013FCF4 0x22222222 0x0013FCF8 0x61616161

解析：

1.对于x86，栈的增长方向是从大地址到小地址

2.对于函数调用，参数的入栈顺序是从右向左

3.函数调用入栈顺序是  右边参数-->左边参数-->函数返回地址

0x0013FCF0    0x0013FCF8

0x0013FCF4    0x22222222

0x0013FCF8    0x61616161



递归先序遍历一个节点为n，深度为d的二叉树，需要栈空间的大小为\_\_\_\_\_\_。B

A.O(n)

B.O(d)

C.O(log(n))

D.O(n\*log(n))

解析：

因为二叉树并不一定是平衡的，也就是深度d！=logn，有可能d>>logn。。所以栈大小应该是O（d）

int BiTreeDepth( BiTree \*T )

{

int dep1 = 0, dep2 = 0;

if ( T == NULL )

return 0;

else

{

dep1 = BiTreeDepth( T->Lchild );

dep2 = BiTreeDepth( T->Rchild );

if ( dep1 > dep2 )

return dep1 + 1;

else

return dep2 + 1;

}

}



用俩个栈模拟实现一个队列，如果栈的容量分别是O和P(O>P)，那么模拟实现的队列最大容量是多少？ C

A.O+P

B.2O+1

C.2P+1

D.2O-1

解析：

到 设 我 们 只 有 有 3 个 元 素 MBhabc 那 么 两 栈 实 现 队 列 如 下 图 所 示 
入 栈 顺 abc 
栈 0 出 栈 ， EP 
入 栈 顺 序 为 c b a 
为 a b c 
那 么 现 在 到 设 我 们 柱 P 的 奢 0 大 小 为 p ， 最 理 想 情 况 下 ， 会 有 下 图 所 示 
2P 
2 
我 们 将 栈 P 出 栈 ， 在 将 2P 2P 一 尹 2 p 个 元 素 ， 依 次 出 栈 丙 压 入 栈 已 然 后 栈 O 的 p 凵 元 素 
直 接 出 栈 ， 丙 将 栈 P 中 的 元 素 全 部 出 栈 ， 得 到 1 ， 2 ， 一 p 1 伊 2 ， ． 一 2 2P + 1 的 出 栈 順 序 
符 合 先 进 先 出 的 点 ． 即 从 列 ． 这 是 可 以 看 到 队 列 最 大 容 0 为 20 + 1 



调用函数时，入参及返回地址使用了（）D

A.队列

B.多维数组

C.顺序表

D.栈

解析：

函数调用时，入参和返回地址通常使用栈来进行管理。栈是一种后进先出（LIFO）的数据结构，适合用于实现函数的调用和返回过程。

在函数调用时，函数的参数会被压入调用栈中，然后调用栈会记录当前函数的执行位置，即返回地址。当函数执行完毕后，返回地址会从栈中弹出，程序跳转回调用该函数的位置继续执行。

栈的特性使得它非常适合管理函数调用过程中的入参和返回地址。每次函数调用时，新的入参会被推入栈顶，函数返回时，栈顶的返回地址被弹出。这样可以保证函数调用的顺序和正确的返回顺序。因此，在函数调用时，入参及返回地址使用了栈。



以下那种结构，平均来讲获取任意一个指定值最快？() D

A.二叉排序树

B.队列

C.栈

D.哈希表

解析：

哈希表（Hash Table）是一种使用哈希函数将键映射到存储位置的数据结构。它通过将键映射到存储桶中的索引来快速访问和查找元素。在哈希表中，平均情况下，获取任意一个指定值的时间复杂度是O(1)。这是因为通过哈希函数计算出存储桶的索引，直接访问对应位置即可获取目标值，无需遍历整个数据结构。

相比之下，二叉排序树（Binary Search Tree）、队列（Queue）和栈（Stack）的平均查找时间复杂度要高于O(1)。二叉排序树的平均情况下需要O(log n)的时间复杂度，队列和栈的平均情况下需要O(n)的时间复杂度。因此，从获取任意一个指定值最快的角度考虑，哈希表是更优的选择。



4个圆盘的Hanoi塔,总的移动次数为() C

A.7 B.8 C.15 D.16

解析：

汉诺塔：汉诺塔（又称河内塔）问题是源于印度一个古老传说的益智玩具。大梵天创造世界的时候做了三根金刚石柱子，在一根柱子上从下往上按照大小顺序摞着64片黄金圆盘。大梵天命令婆罗门把圆盘从下面开始按大小顺序重新摆放在另一根柱子上。并且规定，在小圆盘上不能放大圆盘，在三根柱子之间一次只能移动一个圆盘。

设移动n个盘子的汉诺塔问题需要g(n)次移动操作来完成。由展示移动过程算法可知g(n)应是三部分之和。  
(1) 将n个盘上面的n-1个盘子借助C桩从A桩移到B桩上，需g(n-1)次移动；  
(2) 然后将A桩上第n个盘子移到C桩上（1次）；  
(3) 最后，将B桩上的n-1个盘子借助A桩移到C桩上，需g(n-1)次。  
因而有递归关系：  
g(n)=2\*g(n-1)+1  
初始条件(递归出口)：  
g(1)=1  
即 1、3、7、15、31。。。即g(n) = 2^n -1



用数组S[0…n]作为两个栈S1和S2的存储结构，对任何一个栈只有当S全满时才不能做入栈操作。为这两个栈分配空间的最佳方案是 A

A.S1的栈底位置为0，S2的栈底位置为n

B.S1的栈底位置为0，S2的栈底位置为n/2

C.S1的栈底位置为1，S2的栈底位置为n/2

解析：

两个栈的栈底一个在数组第一个元素，朝着数组正方向增长 另一个在数组最后一个元素，朝着数组索引减小的方向增长。 当两个栈的栈顶相等是，表明数组满了，不能继续入栈.



支持子程序调用的数据结构是（ ）。A

A.栈

B.树

C.队列

D.二叉树

解析：

支持子程序调用的常见数据结构是栈。在程序执行过程中，每当一个子程序被调用时，其相关信息（如返回地址、局部变量等）会被保存在栈中。这样可以确保在子程序执行完毕后能够正确返回到调用位置，并且每个子程序都有自己独立的活动记录。

树和二叉树通常用于表示层次性的结构，例如目录树、表达式树等，不直接支持子程序调用。

队列是一种先进先出（FIFO）的数据结构，通常用于缓冲和调度场景，并不是直接用于支持子程序调用。