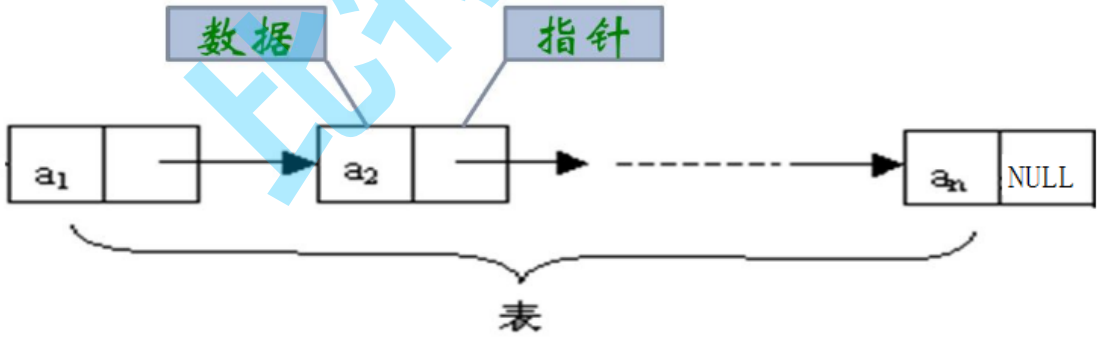
3--线性表之链表

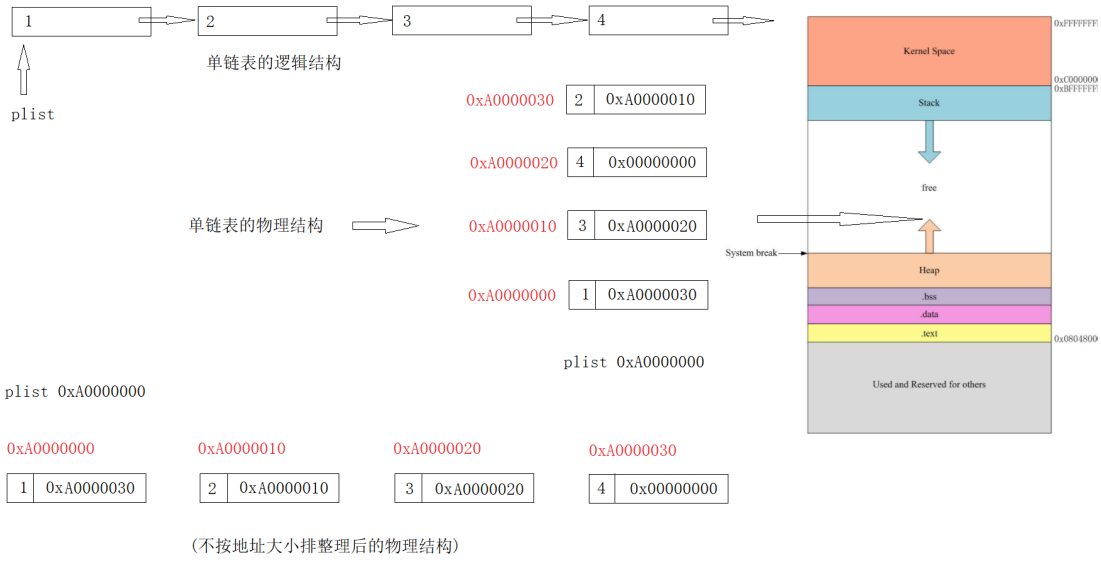
【本节目标】

- **1.链表表示和实现（单链表+双向链表）**
- **2.链表的常见OJ题**
- **3.顺序表和链表的区别和联系**

1.链表表示和实现（单链表+双向链表）

> **顺序表的问题及思考**
>
> 问题：
>
> 1. 中间/头部的插入删除，时间复杂度为O(N)
> 2. 增容需要申请新空间，拷贝数据，释放旧空间。会有不小的消耗。
> 3. 增容一般是呈2倍的增长，势必会有一定的空间浪费。例如当前容量为100，满了以后增容到200，我们再继续插入了5个数据，后面没有数据插入了，那么就浪费了95个数据空间。
>
> 思考：
>
> 如何解决以上问题呢？下面给出了链表的结构来看看。
>
> **1.1 链表的概念及结构**
>
> 概念：链表是一种**物理存储结构上非连续**、非顺序的存储结构，数据元素的**逻辑顺序**是通过链表中的**指针链接**次序实现的 。
>
> 
>

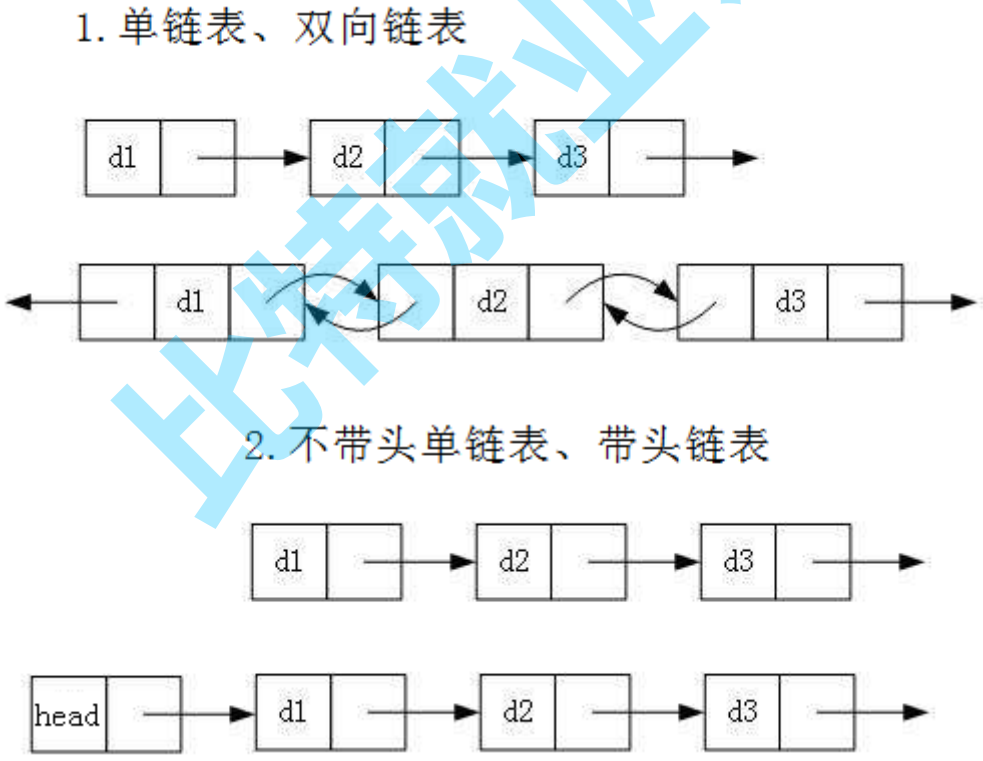
比特就业课



（不按地址大小排整理后的物理结构）

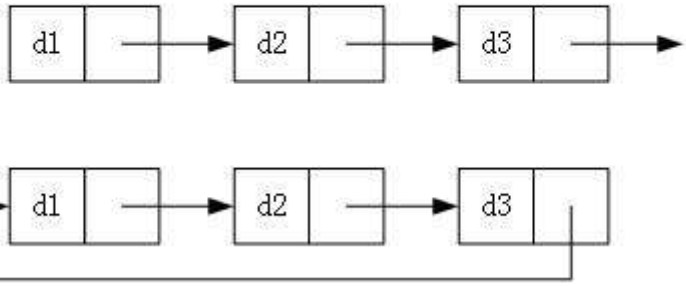
实际中要实现的链表的结构非常多样，以下情况组合起来就有8种链表结构：

1. 单向、双向
2. 带头、不带头
3. 循环、非循环



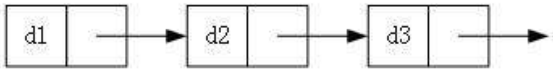
比特就业课

3. 单链表、循环单链表

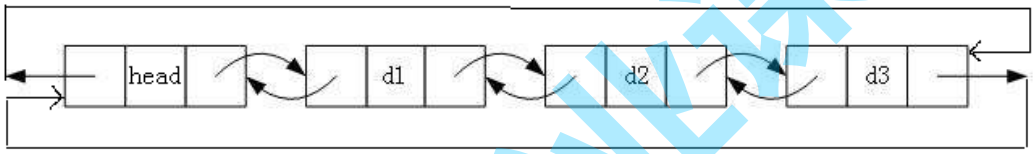


虽然有这么多的链表的结构，但是我们实际中最常用还是两种结构：

无头单向非循环链表



带头双向循环链表



1. 无头单向非循环链表：**结构简单**，一般不会单独用来存数据。实际中更多是作为**其他数据结构的子结构**，如哈希桶、图的邻接表等等。另外这种结构在**笔试面试**中出现很多。
2. 带头双向循环链表：**结构最复杂**，一般用在单独存储数据。实际中使用的链表数据结构，都是带头双向循环链表。另外这个结构虽然结构复杂，但是使用代码实现以后会发现结构会带来很多优势，实现反而简单了，后面我们代码实现了就知道了。

1.2单链表的实现

```
// 1、无头+单向+非循环链表增删查改实现
typedef int SLTDateType;
typedef struct SListNode
{
    SLTDateType data;
    struct SListNode* next;
}SListNode;

// 动态申请一个节点
SListNode* BuySListNode(SLTDateType x);
// 单链表打印
void SListPrint(SListNode* plist);
// 单链表尾插
void SListPushBack(SListNode** pplist, SLTDateType x);
// 单链表的头插
void SListPushFront(SListNode** pplist, SLTDateType x);
// 单链表的尾删
void SListPopBack(SListNode** pplist);
// 单链表头删
void SListPopFront(SListNode** pplist);
// 单链表查找
```

比特就业课

```
SListNode* SListFind(SListNode* plist, SLTDateType x);
// 单链表在pos位置之前插入x
void SListInsert(SListNode** pplist, SListNode* pos, SLTDateType x);
// 单链表删除pos位置的值
void SListErase(SListNode** pplist, SListNode* pos);
```

1.3 双向链表的表示和实现

```
// 2、带头+双向+循环链表增删查改实现
typedef int LTDataType;
typedef struct ListNode
{
    LTDataType _data;
    struct ListNode* _next;
    struct ListNode* _prev;
}ListNode;

// 创建返回链表的头结点.
ListNode* ListCreate();
// 双向链表销毁
void ListDestory(ListNode* plist);
// 双向链表打印
void ListPrint(ListNode* plist);
// 双向链表尾插
void ListPushBack(ListNode* plist, LTDataType x);
// 双向链表尾删
void ListPopBack(ListNode* plist);
// 双向链表头插
void ListPushFront(ListNode* plist, LTDataType x);
// 双向链表头删
void ListPopFront(ListNode* plist);
// 双向链表查找
ListNode* ListFind(ListNode* plist, LTDataType x);
// 双向链表在pos的前面进行插入
void ListInsert(ListNode* pos, LTDataType x);
// 双向链表删除pos位置的节点
void ListErase(ListNode* pos);
```

2.链表的常见OJ题

1.反转单链表：https://leetcode-cn.com/problems/reverse-linked-list/

反转一个单链表。

示例:

```
输入: 1->2->3->4->5->NULL
输出: 5->4->3->2->1->NULL
```

思路一：三个指针翻转

n1 n2 n3

1->2->3->4->5->NULL

思路二：取节点头插到新链表

cur

1->2->3->4->5->NULL

newhead

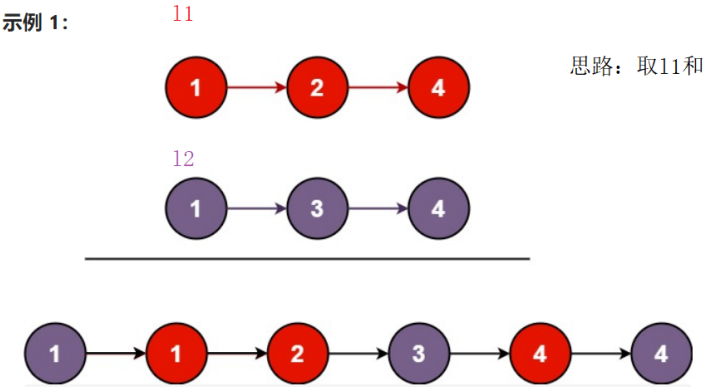
NULL

2.链表的中间结点：https://leetcode-cn.com/problems/middle-of-the-linked-list/

3.合并两个有序链表：https://leetcode-cn.com/problems/merge-two-sorted-lists/

比特就业课

示例 1：

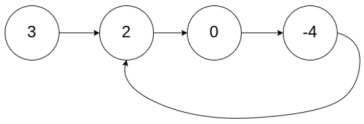


思路：取11和12中小的节点尾插到新联表

4.判断链表是否有环？：https://leetcode-cn.com/problems/linked-list-cycle/

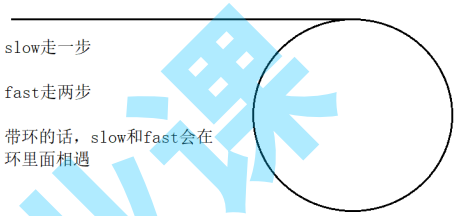
5.求环形链表的入口点？：https://leetcode-cn.com/problems/linked-list-cycle-ii/

示例 1：




```
输入：head = [3,2,0,-4], pos = 1
输出：true
解释：链表中有一个环，其尾部连接到第二个节点。
```

带环链表的抽象图



附加问题1：为什么slow走一步，fast走两步，他们一定会在环里面相遇？会不会永远追不上？请证明！
如果slow走一步，fast走n步(n = 3、4、5...)，他们会不会在环里面相遇？

附加问题2：请分析证明，如何求环的长度，以及环的入口点

3.顺序表和链表的区别和联系

顺序表：

优点：

空间连续、支持随机访问

缺点：

1. 中间或前面部分的插入删除时间复杂度O(N)
2. 2.增容的代价比较大。

链表：

缺点：

以节点为单位存储，不支持随机访问

优点：

1. 任意位置插入删除时间复杂度为O(1)
2. 没有增容消耗，按需申请节点空间，不用了直接释放。

4.顺序表和链表概念选择题（附加作业）

概念选择题：

比特就业课

1.在一个长度为n的顺序表中删除第i个元素，要移动_____个元素。如果要在第i个元素前插入一个元素，要后移_____个元素。
A n-i，n-i+1
B n-i+1，n-i
C n-i，n-i
D n-i+1，n-i+1

2.取顺序表的第i个元素的时间同i的大小有关()
A 对
B 错

3.在一个具有 n 个结点的有序单链表中插入一个新结点并仍然保持有序的时间复杂度是 。
A O（1）
B O（n）
C O（n2）
D O（nlog2n）

4.下列关于线性链表的叙述中，正确的是（ ）。
A 各数据结点的存储空间可以不连续，但它们的存储顺序与逻辑顺序必须一致
B 各数据结点的存储顺序与逻辑顺序可以不一致，但它们的存储空间必须连续
C 进行插入与删除时，不需要移动表中的元素
D 以上说法均不正确

5.设一个链表最常用的操作是在末尾插入结点和删除尾结点，则选用（）最节省时间。
A 单链表
B 单循环链表
C 带尾指针的单循环链表
D 带头结点的双循环链表

6.链表不具有的特点是（）。
A 插入、删除不需要移动元素
B 不必事先估计存储空间
C 可随机访问任一元素
D 所需空间与线性表长度成正比

7.在一个单链表中，若删除 P 所指结点的后续结点，则执行?
A p = p->next; p->next = p->next->next;
B p->next = p->next;
C p->next = p->next->next;
D p = p->next->next

8.一个单向链表队列中有一个指针p，现要将指针r插入到p之后，该进行的操作是_____。
A p->next=p->next->next
B r->next=p;p->next=r->next
C r->next=p->next;p->next=r
D r=p->next;p->next=r->next
E r->next=p;p->next=r
F p=p->next->next

答案:

比特就业课

1.A
2.B
3.B
4.C
5.D
6.C
7.C
8.C

比特就业课