

数据结构与算法——总结

2024 Spring, Compiled by 邓锦文, 物理学院

Part 1: 线性表

线性表 (*List*) 的定义: 零个或多个数据元素的有限序列。

线性表的数据集合为 $\{a_1, a_2, \dots, a_n\}$, 该序列有唯一的头元素和尾元素, 除了头元素外, 每个元素都有唯一的前驱元素, 除了尾元素外, 每个元素都有唯一的后继元素。

线性表中的元素属于相同的数据类型, 即每个元素所占的空间相同。

框架:

线性表

顺序存储

链式存储

顺序表

单链表

双链表

循环链表

(1)

一、顺序表

python中的顺序表就是列表, 元素在内存中连续存放, 每个元素都有唯一序号(下标), 且根据序号访问(包括读取和修改)元素的时间复杂度是 $O(1)$ 的(随机访问)。

```
1 class SequentialList:
2     def __init__(self, n):
3         self.data = list(range(n))
4
5     def is_empty(self):
6         return len(self.data) == 0
7
8     def length(self):
9         return len(self.data)
10
11    def append(self, item):
12        self.data.append(item)
13
14    def insert(self, index, item):
15        self.data.insert(index, item)
16
17    def delete(self, index):
18        if 0 <= index < len(self.data):
19            del self.data[index]
20        else:
21            return IndexError('Index out of range')
22
23    def get(self, index):
24        if 0 <= index < len(self.data):
```

```

25         return self.data[index]
26     else:
27         return IndexError('Index out of range')
28
29     def set(self, index, target):
30         if 0 <= index < len(self.data):
31             self.data[index] = target
32         else:
33             return IndexError('Index out of range')
34
35     def display(self):
36         print(self.data)
37
38 lst = SequentialList(n)
39

```

关于线性表的时间复杂度：

生成、求表中元素个数、表尾添加/删除元素、返回/修改对应下标元素，均为 $O(1)$ ；

而查找、删除、插入元素，均为 $O(n)$ 。

线性表的优缺点：

优点：1、无须为表中元素之间的逻辑关系而增加额外的存储空间；

2、可以快速的存取表中任一位置的元素。

缺点：1、插入和删除操作需要移动大量元素；

2、当线性表长度较大时，难以确定存储空间的容量；

3、造成存储空间的“碎片”。

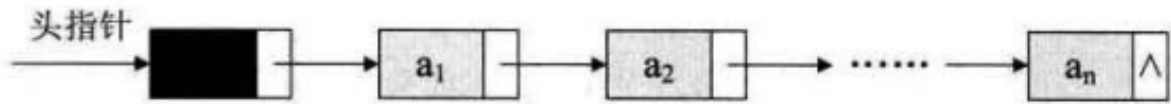
二、链表

1、单链表

在链式结构中，除了要存储数据元素的信息外，还要存储它的后继元素的存储地址。

因此，为了表示每个数据元素 a_i 与其直接后继元素 a_{i+1} 之间的逻辑关系，对数据 a_i 来说，除了存储其本身的信息之外，还需要存储一个指示其直接后继的信息（即直接后继的存储位置）。我们把存储数据元素信息的域称为数据域，把存储直接后继位置的域称为指针域。指针域中存储的信息称做指针或链。这两部分信息组成数据元素 a_i 的存储映像，称为结点（*Node*）。

我们把链表中第一个结点的存储位置叫做头指针。有时为了方便对链表进行操作，会在单链表的第一个结点前附设一个节点，称为头结点，此时头指针指向的结点就是头结点。



空链表，头结点的直接后继为空。



```

1  class LinkList:
2      class Node:
3          def __init__(self, data, next=None):
4              self.data, self.next = data, next
5
6      def __init__(self):
7          self.head = self.tail = LinkList.Node(None, None)
8          self.size = 0
9
10     def print(self):
11         ptr = self.head
12         while ptr is not None:
13             print(ptr.data, end=',')
14             ptr = ptr.next
15
16     def insert(self, p, data):
17         nd = LinkList.Node(data, None)
18         if self.tail is p:
19             self.tail = nd
20         nd.next = p.next
21         p.next = nd
22         self.size += 1
23
24     def delete(self, p):
25         if self.tail is p.next:
26             self.tail = p
27         p.next = p.next.next
28         self.size -= 1
29
30     def popFront(self):
31         if self.head is None:
32             raise Exception("Popping front for Empty link list.")
33         else:
34             self.head = self.head.next
35             self.size -= 1
36             if self.size == 0:
37                 self.head = self.tail = None
38
39     def pushFront(self, data):
40         nd = LinkList.Node(data, self.head)
41         self.head = nd

```

```

42         self.size += 1
43         if self.size == 1:
44             self.tail = nd
45
46     def pushBack(self, data):
47         if self.size == 0:
48             self.pushFront(data)
49         else:
50             self.insert(self.tail, data)
51
52     def clear(self):
53         self.head = self.tail = None
54         self.size = 0
55
56     def __iter__(self):
57         self.ptr = self.head
58         return self
59
60     def __next__(self):
61         if self.ptr is None:
62             raise StopIteration()
63         else:
64             data = self.ptr.data
65             self.ptr = self.ptr.next
66             return data
67

```

2、双链表

双向链表(*Double Linked List*)是在单链表的每个结点中, 再设置一个指向其前驱结点的指针域。所以在双向链表中的结点都有两个指针域, 一个指向直接后继, 另一个指向直接前驱。

```

1  class DoubleLinkedList:
2      class Node:
3          def __init__(self, data, prev=None, next=None):
4              self.data, self.prev, self.next = data, prev, next
5
6      class Iterator:
7          def __init__(self, p):
8              self.ptr = p
9
10         def get(self):
11             return self.ptr.data
12
13         def set(self, data):
14             self.ptr.data = data
15
16         def __iter__(self):

```

```

17         self.ptr = self.ptr.next
18         if self.ptr is None:
19             return None
20         else:
21             return DoubleLinkedList.Iterator(self.ptr)
22
23     def prev(self):
24         self.ptr = self.ptr.prev
25         return DoubleLinkedList.Iterator(self.ptr)
26
27     def __init__(self):
28         self.head = self.tail = DoubleLinkedList.Node(None, None, None)
29         self.size = 0
30
31     def _insert(self, p, data):
32         nd = DoubleLinkedList.Node(data, p, p.next)
33         if self.tail is p:
34             self.tail = nd
35         if p.next:
36             p.next.prev = nd
37         p.next = nd
38         self.size += 1
39
40     def _delete(self, p):
41         if self.size == 0 or p is self.head:
42             return Exception("Illegal deleting.")
43         else:
44             p.prev.next = p.next
45             if p.next:
46                 p.next.prev = p.prev
47             if self.tail is p:
48                 self.tail = p.prev
49             self.size -= 1
50
51     def clear(self):
52         self.tail = self.head
53         self.head.next = self.head.prev = None
54         self.size = 0
55
56     def begin(self):
57         return DoubleLinkedList.Iterator(self.head.next)
58
59     def end(self):
60         return None
61
62     def insert(self, i, data):
63         self._insert(i.ptr, data)
64
65     def delete(self, i):
66         self._delete(i.ptr)
67
68     def pushFront(self, data):

```

```

69         self._insert(self.head, data)
70
71     def pushBack(self, data):
72         self._insert(self.tail, data)
73
74     def popFront(self):
75         self._delete(self.head.next)
76
77     def popBack(self):
78         self._delete(self.tail)
79
80     def __iter__(self):
81         self.ptr = self.head.next
82         return self
83
84     def __next__(self):
85         if self.ptr is None:
86             raise StopIteration()
87         else:
88             data = self.ptr.data
89             self.ptr = self.ptr.next
90             return data
91
92     def find(self, val):
93         ptr = self.head.next
94         while ptr is not None:
95             if ptr.data == val:
96                 return DoubleLinkedList.Iterator(ptr)
97             ptr = ptr.next
98         return self.end()
99
100    def printList(self):
101        ptr = self.head.next
102        while ptr is not None:
103            print(ptr.data, end=', ')
104            ptr = ptr.next
105

```

3、循环链表

将单链表中终端节点的指针端由空指针改为指向头结点，就使整个单链表形成一个环，这种头尾相接的单链表称为单循环链表，简称循环链表。

然而这样会导致访问最后一个结点时需要 $O(n)$ 的时间，所以我们可以写出仅设尾指针的循环链表。

```

1 class CircleLinkedList:
2     class Node:
3         def __init__(self, data, next=None):

```

```

4         self.data, self.next = data, next
5
6     def __init__(self):
7         self.tail = None
8         self.size = 0
9
10    def is_empty(self):
11        return self.size == 0
12
13    def pushFront(self, data):
14        nd = CircleLinkedList.Node(data)
15        if self.is_empty():
16            self.tail = nd
17            nd.next = self.tail
18        else:
19            nd.next = self.tail.next
20            self.tail.next = nd
21        self.size += 1
22
23    def pushBack(self, data):
24        self.pushFront(data)
25        self.tail = self.tail.next
26
27    def popFront(self):
28        if self.is_empty():
29            return None
30        else:
31            nd = self.tail.next
32            self.size -= 1
33            if self.size == 0:
34                self.tail = None
35            else:
36                self.tail.next = nd.next
37        return nd.data
38
39    def popBack(self):
40        if self.is_empty():
41            return None
42        else:
43            nd = self.tail.next
44            while nd.next != self.tail:
45                nd = nd.next
46            data = self.tail.data
47            nd.next = self.tail.next
48            self.tail = nd
49            return data
50
51    def printList(self):
52        if self.is_empty():
53            print('Empty!')
54        else:
55            ptr = self.tail.next

```

```

56         while True:
57             print(ptr.data, end=', ')
58             if ptr == self.tail:
59                 break
60             ptr = ptr.next
61         print()
62

```

三、一些题目

1、约瑟夫问题

<http://cs101.openjudge.cn/practice/02746/>

```

1  class Node:
2      def __init__(self, data):
3          self.data = data
4          self.next = None
5
6  class CircleLinkedList:
7      def __init__(self):
8          self.tail = None
9          self.size = 0
10
11     def append(self, data):
12         nd = Node(data)
13         if self.size == 0:
14             self.tail = nd
15             nd.next = self.tail
16         else:
17             nd.next = self.tail.next
18             self.tail.next = nd
19             self.tail = self.tail.next
20         self.size += 1
21
22     def remove(self, node):
23         current = self.tail.next
24         prev = self.tail
25         while current != self.tail:
26             if current.data == node.data:
27                 prev.next = current.next
28                 self.size -= 1
29                 break
30             prev = current
31             current = current.next
32         if current == self.tail and current.data == node.data:
33             prev.next = current.next
34             self.tail = prev

```



```

35         self.size -= 1
36
37     def get(self, m):
38         current = self.tail.next
39         while self.size > 1:
40             for _ in range(m-1):
41                 current = current.next
42             nd = current.next
43             self.remove(current)
44             current = nd
45         return self.tail.data
46
47 while True:
48     n, m = map(int, input().split())
49     if n == m == 0:
50         break
51     monkey = CircleLinkList()
52     for i in range(1, n+1):
53         monkey.append(i)
54     win = monkey.get(m)
55     print(win)
56

```

2、单向链表

<https://www.luogu.com.cn/problem/B3631>

这道题如果按照最初给的写法写，会超时，是因为每次无论是插入、查询还是删除都需要先找到相应的结点。所以可以使用字典，能够节省大量时间。

```

1  class Node:
2      def __init__(self, data):
3          self.data = data
4          self.next = None
5
6  class LinkList:
7      def __init__(self):
8          self.head = Node(1)
9          self.map = {1: self.head}
10
11     def insert(self, x, y):
12         nx = self.map.get(x)
13         ny = Node(y)
14         ny.next = nx.next
15         nx.next = ny
16         self.map[y] = ny
17
18     def check(self, x):
19         nx = self.map.get(x)
20         if nx.next is None:

```

```

21         print(0)
22         return
23     print(nx.next.data)
24
25     def delete(self, x):
26         nx = self.map.get(x)
27         tmp = nx.next
28         nx.next = tmp.next
29         del self.map[tmp.data]
30
31     lst = LinkList()
32     for _ in range(int(input())):
33         nums = list(map(int, input().split()))
34         if nums[0] == 1:
35             x, y = nums[1:]
36             lst.insert(x, y)
37         elif nums[0] == 2:
38             x = nums[1]
39             lst.check(x)
40         else:
41             x = nums[1]
42             lst.delete(x)
43

```

3、删除链表元素

<http://dsbpython.openjudge.cn/dspythonbook/P0020/>

可以练习循环链表的基本知识（只是缩进有点.....）。

```

1 class Node:
2     def __init__(self, data, next=None):
3         self.data, self.next = data, next
4 class LinkList: #循环链表
5     def __init__(self):
6         self.tail = None
7         self.size = 0
8     def isEmpty(self):
9         return self.size == 0
10    def pushFront(self,data):
11        nd = Node(data)
12        if self.tail == None:
13            self.tail = nd
14            nd.next = self.tail
15        else:
16            nd.next = self.tail.next
17            self.tail.next = nd
18        self.size += 1
19    def pushBack(self,data):
20        self.pushFront(data)

```

```

21     self.tail = self.tail.next
22 def popFront(self):
23     if self.size == 0:
24         return None
25     else:
26         nd = self.tail.next
27         self.size -= 1
28         if self.size == 0:
29             self.tail = None
30         else:
31             self.tail.next = nd.next
32     return nd.data
33 def printList(self):
34     if self.size > 0:
35         ptr = self.tail.next
36         while True:
37             print(ptr.data,end = " ")
38             if ptr == self.tail:
39                 break
40             ptr = ptr.next
41         print("")
42
43 def remove(self,data):
44     if self.size == 0:
45         return None
46
47     if self.size == 1 and self.tail.data == data:
48         self.tail = None
49         self.size -= 1
50         return True
51
52     current = self.tail.next
53     prev = self.tail
54     while current != self.tail:
55         if current.data == data:
56             prev.next = current.next
57             self.size -= 1
58             return True
59         prev = current
60         current = current.next
61
62     if current.data == data:
63         prev.next = current.next
64         self.tail = prev
65         self.size -= 1
66         return True
67     return False
68
69 t = int(input())
70 for i in range(t):
71     lst = list(map(int,input().split()))
72     lkList = LinkList()

```

```

73     for x in lst:
74         lkList.pushBack(x)
75     lst = list(map(int, input().split()))
76     for a in lst:
77         result = lkList.remove(a)
78         if result == True:
79             lkList.printList()
80         elif result == False:
81             print("NOT FOUND")
82         else:
83             print("EMPTY")
84     print("-----")
85

```

4、双端队列

<http://cs101.openjudge.cn/practice/05902/>

这道题如果用`deque`，可以很快的做出来，但是实际上题目想考察的，是对于链表的书写和运用能力。

```

1  class DoubleQueue:
2      class Node:
3          def __init__(self, val=None):
4              self.val = val
5              self.next = None
6
7      def __init__(self):
8          self.tail = None
9          self.size = 0
10
11     def is_empty(self):
12         return self.size == 0
13
14     def push(self, data):
15         nd = DoubleQueue.Node(data)
16         if self.is_empty():
17             self.tail = nd
18             nd.next = self.tail
19         else:
20             nd.next = self.tail.next
21             self.tail.next = nd
22         self.size += 1
23         self.tail = self.tail.next
24
25     def popFront(self):
26         if self.is_empty():
27             return
28         else:
29             nd = self.tail.next
30             self.size -= 1

```

```

31         if self.size == 0:
32             self.tail = None
33         else:
34             self.tail.next = nd.next
35         return
36
37     def popBack(self):
38         if self.is_empty():
39             return
40         else:
41             nd = self.tail.next
42             while nd.next != self.tail:
43                 nd = nd.next
44             nd.next = self.tail.next
45             self.tail = nd
46             self.size -= 1
47         return
48
49     def Print(self):
50         if self.is_empty():
51             print('NULL')
52             return
53         else:
54             nd = self.tail.next
55             while True:
56                 print(nd.val, end=' ')
57                 if nd == self.tail:
58                     break
59                 nd = nd.next
60             print()
61             return
62
63 for _ in range(int(input())):
64     deque = DoubleQueue()
65     for _ in range(int(input())):
66         t, c = map(int, input().split())
67         if t == 1:
68             deque.push(c)
69         else:
70             if c:
71                 deque.popBack()
72             else:
73                 deque.popFront()
74     deque.Print()
75

```

四、一些基本的写法

1、反转链表

改变链表的顺序，使其反向，可以通过迭代或递归实现。

```
1 class Node:
2     def __init__(self):
3         self.val = 0
4         self.next = None
5
6 # 迭代
7 def ReverseList(head: Node) -> Node:
8     prev = None
9     current = head
10    while current is not None:
11        tmp = current.next
12        current.next = prev
13        prev = current
14        current = tmp
15    return prev
16
17 # 递归
18 def reverse(head: Node) -> Node:
19     if head.next is None:
20         return head
21     new_head = reverse(head.next)
22     head.next = None
23     new_head.next = head
24     return new_head
25
```

2、检测环

判断链表是否含有环，快慢指针（*Floyd* 的检测循环算法）是解决这个问题的一个著名方法。

```
1 def has_loop(head: Node) -> bool:
2     slow, fast = head, head
3     while fast and fast.next:
4         slow = slow.next
5         fast = fast.next.next
6         if slow == fast:
7             return True
8     return False
9
```

3、合并两个排序链表

给定两个已排序的链表，要求合并它们并保持排序。这个问题可以用迭代或递归解决。

P.S : 这已经涉及到部分归并排序的内容了。

```
1 def merge(p: Node, q: Node):
2     if p is None or q is None:
3         return p or q
4     if p.val < q.val:
5         p.next = merge(p.next, q)
6         return p
7     else:
8         q.next = merge(p, q.next)
9         return q
10
```

4、删除倒数第n个元素

设置两个指针，一个先走*n*步，之后一起走，先走的到达末尾的时候，另一个就在对应位置了。

```
1 def remove(head: Node, n: int) -> Node:
2     dummy = Node()
3     dummy.next = head
4     slow, fast = dummy, dummy
5     for _ in range(n+1):
6         fast = fast.next
7     while fast:
8         slow = slow.next
9         fast = fast.next
10    slow.next = slow.next.next
11    return dummy.next
12
```

5、查找中间元素

感觉和上一条差不多.....如果设置了*self.size*的话就更好处理了，总之可以使用快慢指针。

```
1 def middle(head: Node) -> Node:
2     slow, fast = head, head
3     while fast.next is not None:
4         slow = slow.next
5         fast = fast.next.next
6     return slow
7
```

6、判断是否回文链表

其实就是先找中间点，然后反转，最后进行判断即可。

```

1  def check(head: Node) -> bool:
2      slow, fast = head, head
3      while fast is not None and fast.next is not None:
4          fast = fast.next.next
5          slow = slow.next
6      prev = None
7      while slow:
8          tmp = slow.next
9          slow.next = prev
10         prev = slow
11         slow = tmp
12     while prev is not None:
13         if head.val != prev.val:
14             return False
15         head = head.next
16         prev = prev.next
17     return True
18

```

7、移除链表元素

删除链表中所有等于给定的结点，要求一次遍历解决。

```

1  def remove(head: Node, data: int) -> Node:
2      dummy = Node()
3      dummy.next = head
4      current = dummy
5      while current.next is not None:
6          if current.next.val == data:
7              current.next = current.next.next
8          else:
9              current = current.next
10     return dummy.next
11

```

8、分割链表

根据给定值 x 分割链表，使得所有小于 x 的节点都位于大于或等于 x 的节点之前，保持原有元素的相对顺序。

```

1  def partition(head: Node, x: int) -> Node:
2      before_node = Node()
3      before = before_node
4      after_node = Node()
5      after = after_node
6      while head is not None:
7          if head.val < x:
8              before.next = head

```



```

9         before = before.next
10    else:
11        after.next = head
12        after = after.next
13        head = head.next
14    after.next = None
15    before.next = after_node.next
16    return before_node.next
17

```

9、旋转链表

给定一个链表，每个元素向右移动 k 个位置。可以先讲单链表连通，形成环，之后再断开即可。

```

1  def rotate(head: Node, k: int) -> Node:
2      if head.next is None or k == 0:
3          return head
4      length = 1
5      tail = head
6      while tail.next is not None:
7          tail = tail.next
8          length += 1
9      tail.next = head
10     new_tail = head
11     for _ in range(length-k % length-1):
12         new_tail = new_tail.next
13     new_head = new_tail.next
14     new_tail.next = None
15     return new_head
16

```

Part 2: 排序

一、冒泡排序(*Bubble Sort*)

冒泡排序是最简单的排序算法，它通过不断交换相邻元素以实现正确的排序结果。

时间复杂度： $O(n^2)$ ；空间复杂度： $O(1)$ 。

冒泡排序是一种原地排序算法，无需额外空间，是稳定的。

```

1  def bubble_sort(arr):
2      n = len(arr)
3      for i in range(n):
4          swapped = False
5          for j in range(n-i-1):
6              if arr[j] > arr[j+1]:
7                  arr[j], arr[j+1] = arr[j+1], arr[j]
8                  swapped = True
9          if not swapped:
10             break
11

```

二、选择排序(*Selection Sort*)

选择排序是一种简单且高效的排序算法，其工作原理是从列表的未排序部分反复选择最小(或最大)元素，并将其移动到列表的已排序部分。

时间复杂度： $O(n^2)$ ；空间复杂度： $O(1)$ 。

选择排序同样是一种原地排序算法，无需额外空间，但它在小的数据集下相对高效，在极端情况下会具有较大的时间复杂度，是不稳定的。

```

1  def selection_sort(arr):
2      for p in range(len(arr)-1, 0, -1):
3          position = 0
4          for location in range(1, p+1):
5              if arr[location] > arr[position]:
6                  position = location
7          if p != position:
8              arr[p], arr[position] = arr[position], arr[p]
9

```

三、快速排序(*Quick Sort*)

快速排序是一种基于分治算法的排序算法，它选择一个元素作为基准，并通过将基准放置在已排序数组中的正确位置来围绕所选择的基准对给定数组进行分区。

时间复杂度：最好时为 $O(n\log n)$ ，最差时为 $O(n^2)$ ；空间复杂度：考虑递归堆栈，为 $O(n)$ ，不考虑则为 $O(1)$ 。

快速排序相对更适用于大数据集，在某些极端情况下会显现出较差的时间复杂度，是不稳定的。

P. S：似乎二叉搜索树的建树再加上中序遍历，就可以获得同样的结果。

```

1  def quick_sort(arr, left, right):
2      if left < right:
3          position = partition(arr, left, right)
4          quick_sort(arr, left, position-1)
5          quick_sort(arr, position+1, right)
6  def partition(arr, left, right):
7      i = left
8      j = right-1
9      pivot = arr[right]
10     while i <= j:
11         while i <= right and arr[i] < pivot:
12             i += 1
13         while j >= left and arr[j] >= pivot:
14             j -= 1
15         if i < j:
16             arr[i], arr[j] = arr[j], arr[i]
17     if arr[i] > pivot:
18         arr[i], arr[right] = arr[right], arr[i]
19     return i
20

```

四、归并排序(*Merge Sort*)

归并排序作为一种排序算法，其原理是将数组划分为更小的子数组，对每个子数组进行排序，然后将排序后的子数组合并在一起，形成最终的排序数组。（和快速排序同样的分治思想）

时间复杂度： $O(n\log n)$ ；空间复杂度： $O(n)$ 。

归并排序是一种天然的可并行化算法，而且稳定，因此特别适合用来处理大数据集，但它需要额外空间。

```

1  def merge_sort(arr):
2      if len(arr) > 1:
3          mid = len(arr) // 2
4          left = arr[:mid]
5          right = arr[mid:]
6          merge_sort(left)
7          merge_sort(right)
8          i, j, k = 0, 0, 0
9          while i < len(left) and j < len(right):
10             if left[i] <= right[j]:
11                 arr[k] = left[i]
12                 i += 1
13             else:
14                 arr[k] = right[j]
15                 j += 1
16             k += 1
17         while i < len(left):
18             arr[k] = left[i]

```

```

19         i += 1
20         k += 1
21     while j > len(right):
22         arr[k] = right[j]
23         j += 1
24         k += 1
25

```

这里附两道题，求逆序数的，之前一直用树状数组.....

http://cs101.openjudge.cn/2024sp_routine/02299/

```

1  def merge(nums, left, mid, right):
2      l_nums = nums[left:mid+1]
3      r_nums = nums[mid+1:right+1]
4      i, j = 0, 0
5      count = 0
6      for k in range(left, right+1):
7          if i < len(l_nums) and (j >= len(r_nums) or l_nums[i] < r_nums[j]):
8              nums[k] = l_nums[i]
9              i += 1
10         else:
11             nums[k] = r_nums[j]
12             j += 1
13             count += len(l_nums) - i
14     return count
15
16 def merge_sort(nums, left, right):
17     if left < right:
18         mid = (left+right)//2
19         count = merge_sort(nums, left, mid)
20         count += merge_sort(nums, mid+1, right)
21         count += merge(nums, left, mid, right)
22     return count
23     return 0
24
25 while True:
26     n = int(input())
27     if n == 0:
28         break
29     nums = [int(input()) for _ in range(n)]
30     print(merge_sort(nums, 0, n-1))

```

http://cs101.openjudge.cn/2024sp_routine/09201/

```

1  def merge(nums, left, mid, right):
2      l_nums = nums[left: mid+1]
3      r_nums = nums[mid+1: right+1]

```

```

4     i, j = 0, 0
5     count = 0
6     for k in range(left, right+1):
7         if i < len(l_nums) and (j >= len(r_nums) or l_nums[i] < r_nums[j]):
8             nums[k] = l_nums[i]
9             i += 1
10        else:
11            nums[k] = r_nums[j]
12            count += len(l_nums)-i
13            j += 1
14    return count
15
16    def merge_sort(nums, left, right):
17        if left < right:
18            mid = (left+right)//2
19            count = merge_sort(nums, left, mid)
20            count += merge_sort(nums, mid+1, right)
21            count += merge(nums, left, mid, right)
22            return count
23    return 0
24
25    n = int(input())
26    nums = list(map(int, input().split()))[::-1]
27    print(merge_sort(nums, 0, n-1))
28

```

其实用树状数组可以过（上面那一道会MLE）：

```

1    def low_bit(x):
2        return x & -x
3    n = int(input())
4    nums = list(map(int, input().split()))
5    my_dict = dict(zip(sorted(nums), list(range(1, n+1))))
6    tree = [0 for _ in range(n+1)]
7    ans = 0
8    for i in range(n):
9        num = my_dict[nums[i]]
10       x = num - 1
11       while x:
12           ans += tree[x]
13           x -= low_bit(x)
14       while num <= n:
15           tree[num] += 1
16           num += low_bit(num)
17    print(ans)
18

```

五、插入排序(*Insertion Sort*)

插入排序是一种基本的排序算法，其思想在于通过先前的已排序的数组得到目标插入元素的插入位置，从而达到不断排序的目的。

时间复杂度： $O(n^2)$ ；空间复杂度： $O(1)$ 。

插入排序是一种稳定的原地排序算法。

```
1 def insertion_sort(arr):
2     for i in range(1, len(arr)):
3         key = arr[i]
4         j = i-1
5         while j >= 0 and key < arr[j]:
6             arr[j+1] = arr[j]
7             j -= 1
8         arr[j+1] = key
9
```

六、希尔排序(*Shell Sort*)

希尔排序可以看作是插入排序的变种，也就相当于可以交换远项。

时间复杂度：（最差） $O(n^2)$ ；空间复杂度： $O(1)$ 。

希尔排序的时间复杂度取决于算法的对象，是不稳定的。

```
1 def shell_sort(arr, n):
2     gap = n // 2
3     while gap > 0:
4         j = gap
5         while j < n:
6             i = j - gap
7             while i >= 0:
8                 if arr[i+gap] > arr[i]:
9                     break
10                else:
11                    arr[i+gap], arr[i] = arr[i], arr[i+gap]
12                i -= gap
13            j += 1
14        gap //= 2
15
```

七、堆排序(*Heap Sort*)

堆排序是一种基于完全二叉树（堆）的排序算法。它通过将待排序的元素构建成一个堆，然后利用堆的性质来实现排序。在堆中，每个结点的值都必须大于等于其子结点的值。

时间复杂度： $O(n\log n)$ ；空间复杂度： $O(1)$ 。

堆排序适合处理大型数据集，采取原地排序，但不稳定，因为可能会交换相同元素。

```
1 def heapify(arr, n, i):
2     largest = i
3     l = 2*i + 1
4     r = 2*i + 2
5     if l < n and arr[l] > arr[largest]:
6         largest = l
7     if r < n and arr[r] > arr[largest]:
8         largest = r
9     if largest != i:
10        arr[i], arr[largest] = arr[largest], arr[i]
11        heapify(arr, n, largest)
12
13 def heapsort(arr):
14     n = len(arr)
15     for i in range(n//2 - 1, -1, -1):
16         heapify(arr, n, i)
17     for i in range(n-1, 0, -1):
18         arr[i], arr[0] = arr[0], arr[i]
19         heapify(arr, i, 0)
20
```

八、计数排序(*Counting Sort*)

计数排序是一种非比较性的整数排序算法。它的基本思想是对于给定的输入序列中的每一个元素 x ，确定该序列中值小于 x 的元素的个数。利用这个信息，可以直接确定 x 在输出序列中的位置，从而实现排序。

时间复杂度： $O(n + k)$ ，其中 n 为序列长度， k 为序列数据范围。空间复杂度也为 $O(n + k)$ 。

计数排序是一种稳定的排序算法，在数据集数值范围较小的情况下表现优越，但倘若数据离散程度较大则效率将会大大降低。

```
1 def counting_sort(arr):
2     max_val = max(arr)
3     min_val = min(arr)
4     counts = [0 for _ in range(max_val-min_val+1)]
5     for num in arr:
6         counts[num-min_val] += 1
7     for i in range(1, max_val-min_val+1):
8         counts[i] += counts[i-1]
9     result = [0 for _ in range(len(arr))]
10    for num in reversed(arr):
```

```

11         result[counts[num-min_val]-1] = num
12         counts[num-min_val] -= 1
13     return result
14

```

九、桶排序(*Bucket Sort*)

桶排序将数组分割成几个部分（桶），然后对每个桶进行排序，最后将所有桶中的元素合并起来得到最终的有序数组。它的基本思想是将待排序数组分割成若干个较小的数组（桶），每个桶再分别排序，最后按照顺序依次取出各个桶中的元素即可。

时间复杂度：理想情况下为 $O(n + k)$ ，极端情况下为 $O(n^2)$ 。空间复杂度同样。

桶排序适合相对均匀稠密的数字序列，在处理小数序列时相对更具有优势（小数更适合用于建立桶的索引与数字之间的映射）。

P.S：桶排序中每个桶使用的排序方法时其它的排序方法，所以桶更适理解作为一种思想，同样的，代码不在此展示。

十、基数排序(*Radix Sort*)

基数排序是一种非比较性的整数排序算法，它根据键值的每位数字来进行排序。基数排序的核心思想是将待排序的元素按照位数切割成不同的数字，然后按照这些位数分别进行排序。这种排序算法属于分配式排序，它的性能取决于桶的使用方法。

时间复杂度： $O(d * (n + k))$ ，空间复杂度： $O(n + k)$ 。

基数排序不依赖于比较操作，适用于整数等固定长度的数据类型，并且在某些情况下具有稳定性。但它需要额外的空间，而且对于位数较大的数据可能不太实用。

```

1  def radix_sort(arr):
2      max_num = max(arr)
3      max_length = len(str(max_num))
4      buckets = [[] for _ in range(10)]
5      for digit in range(max_length):
6          for num in arr:
7              bucket_idx = (num // 10**digit) % 10
8              buckets[bucket_idx].append(num)
9          arr = [num for bucket in buckets for num in bucket]
10         buckets = [[] for _ in range(10)]
11     return arr
12

```


Part 3: 栈和队列

由于在线性表部分已经基本给出了栈和队列的实现形式，因此在这部分不做赘述。个人认为栈和队列更多地是一种思想，蕴含着元素选取顺序的逻辑。

一、栈

1、栈

<https://www.luogu.com.cn/problem/P1044>

以第一个元素举例：

在出栈序列的第一个时，应该先进并且马上出；

在第二个时，应该第二个元素先进然后马上出；

在第三个时，第二、三个元素只要保证先出即可；

.....（依次直至在最后一个）

记 n 个元素对应的种数为 $f(n)$ ，则有公式：

$$f(n) = f(0) * f(n-1) + f(1) * f(n-2) + \dots + f(n-1) * f(0) \quad (2)$$

这其实就是卡特兰数(*Catalan number*)，其公式为：

$$f(n) = \frac{C_{2n}^n}{n+1} \quad (3)$$

接下来就可以直接敲代码AC了。

```
1 import math
2 n = int(input())
3 print(math.comb(2*n, n)/(n+1))
4
```

2、日志分析

<https://www.luogu.com.cn/problem/P1165>

这道题本来想着用堆+懒删除做，但是确实不如使用辅助栈维护最大值。（参考快速堆猪，一样的道理）

```
1 stack = []
2 for _ in range(int(input())):
3     operation = list(map(int, input().split()))
4     if operation[0] == 0:
5         if stack:
```

```

6         stack.append(max(stack[-1], operation[1]))
7     else:
8         stack.append(operation[1])
9 elif operation[0] == 1:
10     if stack:
11         stack.pop()
12 else:
13     if stack:
14         print(stack[-1])
15     else:
16         print(0)
17

```

3、棋盘制作

<https://www.luogu.com.cn/problem/P1169>

这道题有很多方法，悬线法、dp等等，但是个人认为使用单调栈做一做是对于栈的理解帮助不小的（因为我最开始就没理解明白，想了好久hhh）。

```

1  n, m = map(int, input().split())
2  a, b = 1, 1
3  h = [1 for _ in range(m)]
4  stack, up = [], []
5
6  def cut(height, end):
7      global a, b
8      while stack and height <= h[stack[-1]]:
9          k = stack.pop()
10         l = stack[-1] if stack else start
11         a = max(a, h[k]*(end-l))
12         b = max(b, min(h[k], end-l)**2)
13
14 for i in range(n):
15     row = list(map(int, input().split()))
16     left = 0
17     for j, c in enumerate(row):
18         if i:
19             h[j] = 1 if c == up[j] else h[j]+1
20         if not j:
21             start = -1
22         elif c == left:
23             cut(0, j-1)
24             start = j-1
25         else:
26             cut(h[j], j-1)
27             stack.append(j)
28             left = c
29     cut(0, m-1)
30     up = row

```

```
31 print(f'{b}\n{a}')
32
```

4、表达式的转换

<https://www.luogu.com.cn/problem/P1175>

这道题对于栈的应用算是比较经典且常见的（后面的树部分经常用到），同时需要注意幂指数是从右向左算。以及由于数据给的不好，需要对输入作 *strip* 处理。

```
1  def trans(infix):
2      stack = []
3      output = []
4      judge = {'+': 1, '-': 1, '*': 2, '/': 2, '^': 3}
5      for char in infix:
6          if char in judge:
7              while stack and stack[-1] in judge and judge[char] <= judge[stack[-1]]:
8                  if char == '^':
9                      break
10                 output.append(stack.pop())
11                 stack.append(char)
12             elif char == '(':
13                 stack.append(char)
14             elif char == ')':
15                 while stack and stack[-1] != '(':
16                     output.append(stack.pop())
17                 stack.pop()
18             else:
19                 output.append(char)
20     while stack:
21         output.append(stack.pop())
22     return output
23
24 string = list(input().strip())
25 postfix = trans(string)
26 print(*postfix)
27 while len(postfix) > 1:
28     stack = []
29     while True:
30         char = postfix.pop(0)
31         if char in '+-*/':
32             a = stack.pop()
33             b = stack.pop()
34             stack.append(str(int(eval(b+char+a))))
35             break
36         elif char == '^':
37             a = stack.pop()
38             b = stack.pop()
39             stack.append(str(int(b)**int(a)))
40             break
```

```

41         else:
42             stack.append(char)
43         postfix = stack+postfix
44         print(*postfix)
45

```

5、最大数

<https://www.luogu.com.cn/problem/P1198>

这道题考察单调栈，同时用bisect库实现二分查找（很好的工具），如果只是对单调栈进行直接的修改，会超时（大概得80分？），这时也可以使用并查集，但是感觉这么做相对最好想。

```

1  import bisect
2  m, d = map(int, input().split())
3  f, id = [], []
4  t, l = 0, 0
5  for _ in range(m):
6      check, x = input().split()
7      x = int(x)
8      if check == 'A':
9          x = (x+t) % d
10         while f and f[-1] <= x:
11             f.pop()
12             id.pop()
13         f.append(x)
14         id.append(l)
15         l += 1
16     else:
17         t = f[bisect.bisect_left(id, l-x)]
18         print(t)
19

```

6、赛车

<https://www.luogu.com.cn/problem/P3256>

这道题同样考察单调栈，其实主要就是利用栈的特性去构造一种贪心策略，从而得到能够获奖的赛车（得到结果其实是按照它们获得头名的顺序），那么最后的答案就显而易见了。

```

1  class Node:
2      def __init__(self, x, v, id):
3          self.x = x
4          self.v = v
5          self.id = id
6
7  def time(a, b):
8      if a.v == b.v:

```

```

9         return float('inf')
10    return (a.x-b.x)/(b.v-a.v)
11
12    n = int(input())
13    x = list(map(int, input().split()))
14    v = list(map(int, input().split()))
15    races = [Node(0, 0, i) for i in range(1, n+1)]
16    for i in range(n):
17        races[i].x = X[i]
18    for i in range(n):
19        races[i].v = V[i]
20    races.sort(key=lambda x: (x.v, x.x))
21    stack = []
22    stack.append(races[0])
23    for k in range(1, n):
24        p = races[k]
25        while stack:
26            if len(stack) == 1:
27                if p.x > stack[-1].x:
28                    stack.pop()
29            else:
30                break
31        else:
32            if p.x > stack[-1].x:
33                stack.pop()
34            elif time(stack[-1], p) < time(stack[-2], stack[-1]):
35                stack.pop()
36            else:
37                break
38        stack.append(p)
39    nums = [p.id for p in stack]
40    nums.sort()
41    print(len(nums))
42    print(*nums)
43

```

二、队列

1、合唱队形

<https://www.luogu.com.cn/problem/P1091>

先分别求出以对应同学为结尾的长度（向左&向右），再扫一遍即可。（利用了一些单调队列的思想）

```

1    n = int(input())
2    nums = list(map(int, input().split()))
3    a, b = [1 for _ in range(n)], [1 for _ in range(n)]
4    for i in range(n):

```

```

5     for j in range(i):
6         if nums[i] > nums[j]:
7             a[i] = max(a[i], a[j]+1)
8 for i in range(n-1, -1, -1):
9     for j in range(n-1, i, -1):
10        if nums[i] > nums[j]:
11            b[i] = max(b[i], b[j]+1)
12 ans = 0
13 for k in range(n):
14     ans = max(ans, a[k]+b[k]-1)
15 print(n-ans)
16

```

2、机器人搬重物

<https://www.luogu.com.cn/problem/P1126>

典型的bfs，使用双端队列可以降低时间复杂度，同时这里是不建议使用vis的，因为变量太多，用memo去判断会更方便一些，当然这里对于题目的理解也需要注意一下，机器人是不能在边缘行走的（可以认为边缘是墙壁）。

P.S : Openjudge上面也有这道题——Robot: <http://cs101.openjudge.cn/dsapre/01376/>

```

1 from collections import deque
2 def bfs(start, end):
3     if start[0] == end[0] and start[1] == end[1]:
4         return 0
5     queue = deque()
6     queue.append(start)
7     dir = [(0, -1), (-1, 0), (0, 1), (1, 0)]
8     while queue:
9         x, y, d, t = queue.popleft()
10        if memo[x][y][(d+3) % 4] > t+1:
11            memo[x][y][(d+3) % 4] = t+1
12            queue.append([x, y, (d+3) % 4, t+1])
13        if memo[x][y][(d+1) % 4] > t+1:
14            memo[x][y][(d+1) % 4] = t+1
15            queue.append([x, y, (d+1) % 4, t+1])
16        for k in range(1, 4):
17            nx = x + dir[d][0]*k
18            ny = y + dir[d][1]*k
19            if 1 <= nx < n and 1 <= ny < m:
20                if maze[nx][ny]:
21                    break
22                if nx == end[0] and ny == end[1]:
23                    return t+1
24                if memo[nx][ny][d] > t+1:
25                    memo[nx][ny][d] = t+1
26                    queue.append([nx, ny, d, t+1])
27        return -1
28

```

```

29 n, m = map(int, input().split())
30 matrix = [list(map(int, input().split())) for _ in range(n)]
31 maze = [[0 for _ in range(m+1)] for _ in range(n+1)]
32 for i in range(n):
33     for j in range(m):
34         if matrix[i][j]:
35             maze[i][j], maze[i][j+1], maze[i+1][j], maze[i+1][j+1] = 1, 1, 1, 1
36 memo = [[[float('inf') for _ in range(4)] for _ in range(m+1)] for _ in range(n+1)]
37 z = input().split()
38 judge = {'W': 0, 'N': 1, 'E': 2, 'S': 3}
39 d = judge[z[4]]
40 start = [int(z[0]), int(z[1]), d, 0]
41 end = [int(z[2]), int(z[3])]
42 memo[start[0]][start[1]][d] = 0
43 print(bfs(start, end))
44

```

3、逛画展

<https://www.luogu.com.cn/problem/P1638>

这道题和滑动窗口很像，总之就是开一个字典（或者开一个数组也可）记录出现的是哪个画家的作品，扫一遍，保留最短的答案就可以。

```

1  from collections import defaultdict
2  n, m = map(int, input().split())
3  nums = list(map(int, input().split()))
4  my_dict = defaultdict(int)
5  my_dict[nums[0]] = 1
6  k = 1
7  ans = float('inf')
8  left, right, a, b = 0, 0, 0, 0
9  while left <= right < n:
10     if k == m:
11         if ans > right - left + 1:
12             ans = right - left + 1
13             a, b = left, right
14         my_dict[nums[left]] -= 1
15         if my_dict[nums[left]] == 0:
16             k -= 1
17         left += 1
18     else:
19         if right == n-1:
20             break
21         right += 1
22         my_dict[nums[right]] += 1
23         if my_dict[nums[right]] == 1:
24             k += 1
25 print(a+1, b+1)
26

```

4、Saving Tong Monk

<http://cs101.openjudge.cn/practice/04130/>

个人感觉做了这道题之后一般的bfs都不会出问题了😂，思路是用优先队列（*PriorityQueue*或者*heapq*）储存结点，然后获得时间最短的，再拿记忆化搜索就可以（开个三维数组），同时记得使用二进制记录蛇的存活状态。

```
1  from queue import PriorityQueue
2
3  class Node:
4      def __init__(self, x, y, t, k, s):
5          self.x = x
6          self.y = y
7          self.t = t
8          self.k = k
9          self.s = s
10     def __lt__(self, other):
11         return self.t < other.t
12
13     def bfs(maze, n, m):
14         x0, y0, count = 0, 0, 0
15         lst = [[0 for _ in range(n)] for _ in range(n)]
16         for i in range(n):
17             for j in range(n):
18                 if maze[i][j] == 'K':
19                     x0, y0 = i, j
20                 if maze[i][j] == 'S':
21                     lst[i][j] = count
22                     count += 1
23         dir = [(-1, 0), (1, 0), (0, -1), (0, 1)]
24         memo = [[[float('inf')] for _ in range(m+1)] for _ in range(n)] for _ in range(n)]
25         queue = PriorityQueue()
26         queue.put(Node(x0, y0, 0, 0, 0))
27         memo[x0][y0][0] = 0
28         while not queue.empty():
29             node = queue.get()
30             if maze[node.x][node.y] == 'T' and node.k == m:
31                 return node.t
32             for a, b in dir:
33                 nx, ny = node.x+a, node.y+b
34                 if 0 <= nx < n and 0 <= ny < n:
35                     if maze[nx][ny] == '#':
36                         continue
37                     elif maze[nx][ny] == 'S':
38                         if (node.s >> lst[nx][ny]) & 1:
39                             if node.t+1 < memo[nx][ny][node.k]:
40                                 memo[nx][ny][node.k] = node.t+1
41                                 queue.put(Node(nx, ny, node.t+1, node.k, node.s))
42                     else:
```



```

43         if node.t+2 < memo[nx][ny][node.k]:
44             memo[nx][ny][node.k] = node.t+2
45             queue.put(Node(nx, ny, node.t+2, node.k, node.s | (1 <<
lst[nx][ny])))
46         elif maze[nx][ny].isdigit():
47             if int(maze[nx][ny]) == node.k+1:
48                 if node.t+1 < memo[nx][ny][node.k+1]:
49                     memo[nx][ny][node.k+1] = node.t+1
50                     queue.put(Node(nx, ny, node.t+1, node.k+1, node.s))
51             else:
52                 if node.t+1 < memo[nx][ny][node.k]:
53                     memo[nx][ny][node.k] = node.t+1
54                     queue.put(Node(nx, ny, node.t+1, node.k, node.s))
55             else:
56                 if node.t+1 < memo[nx][ny][node.k]:
57                     memo[nx][ny][node.k] = node.t+1
58                     queue.put(Node(nx, ny, node.t+1, node.k, node.s))
59         return 'impossible'
60
61 while True:
62     n, m = map(int, input().split())
63     if n == m == 0:
64         break
65     maze = [list(input()) for _ in range(n)]
66     print(bfs(maze, n, m))
67

```

5、火锅盛宴

<https://www.luogu.com.cn/problem/P4032>

这道题不知道怎样才能减小时间，先把代码放这里吧（TLE版本）。

```

1  import heapq
2  from collections import defaultdict
3
4  def low_bit(x):
5      return x & -x
6
7  def query(x, y):
8      ans = 0
9      while y > x:
10         ans += tree[y]
11         y -= low_bit(y)
12     while x > y:
13         ans += tree[x]
14         x -= low_bit(x)
15     return ans
16
17 def add(x, k):

```

```

18     while x <= n:
19         tree[x] += k
20         x += low_bit(x)
21
22 for _ in range(int(input())):
23     n = int(input())
24     s = list(map(int, input().split()))
25     tree = [0 for _ in range(n+1)]
26     eat = defaultdict(int)
27     food = [[] for _ in range(n)]
28     for _ in range(int(input())):
29         operation = list(map(int, input().split()))
30         if operation[1] == 0:
31             heapq.heappush(food[operation[2]-1], operation[0]+s[operation[2]-1])
32         else:
33             t = operation[0]
34             for k in range(n):
35                 while food[k] and food[k][0] <= t:
36                     heapq.heappop(food[k])
37                     eat[k+1] += 1
38                     add(k+1, 1)
39             if operation[1] == 1:
40                 for k in range(1, n+1):
41                     if eat[k]:
42                         add(k, -1)
43                         eat[k] -= 1
44                         print(k)
45                         break
46             else:
47                 print('Yazid is angry.')
48         elif operation[1] == 2:
49             idx = operation[2]
50             if eat[idx]:
51                 eat[idx] -= 1
52                 add(idx, -1)
53                 print('Succeeded!')
54             else:
55                 if food[idx-1]:
56                     print(food[idx-1][0]-t)
57                 else:
58                     print('YJQQQAQ is angry.')
59         else:
60             l, r = operation[2:]
61             print(query(l-1, r))
62

```

Part 4: 树

植树节开始讲/整理树这一部分，蛮好.....

一、定义

1、节点和边

树由节点及连接节点的边构成。树有以下属性：

- 有一个根节点；
- 除根节点外，其他每个节点都与其唯一的父节点相连；
- 从根节点到其他每个节点都有且仅有一条路径；
- 如果每个节点最多有两个子节点，我们就称这样的树为二叉树。

2、递归

一棵树要么为空，要么由一个根节点和零棵或多棵子树构成，子树本身也是一棵树。每棵子树的根节点通过一条边连到父树的根节点。

二、数据结构

1、二叉堆

二叉堆通过树的特性，由列表实现，每次加入元素时，进行“上浮”操作，而删除元素时则进行“下沉”操作。

以下代码是<http://cs101.openjudge.cn/practice/04078/>的AC代码，自行定义*BinHeap*类实现了堆结构。

```
1 class BinHeap:
2     def __init__(self):
3         self.list = [0]
4         self.size = 0
5
6     def up(self, i):
7         while i // 2 > 0:
8             if self.list[i] < self.list[i // 2]:
9                 tmp = self.list[i // 2]
10                self.list[i // 2] = self.list[i]
11                self.list[i] = tmp
12                i //= 2
13
14     def heappush(self, k):
15         self.list.append(k)
16         self.size += 1
17         self.up(self.size)
18
19     def min(self, i):
20         if i*2+1 > self.size:
```

```

21         return i*2
22     else:
23         if self.list[i*2] < self.list[i*2+1]:
24             return i*2
25         else:
26             return i*2+1
27
28     def down(self, j):
29         while (j*2) <= self.size:
30             t = self.min(j)
31             if self.list[j] > self.list[t]:
32                 tmp = self.list[j]
33                 self.list[j] = self.list[t]
34                 self.list[t] = tmp
35             j = t
36
37     def heappop(self):
38         ans = self.list[1]
39         self.list[1] = self.list[self.size]
40         self.size -= 1
41         self.list.pop()
42         self.down(1)
43         return ans
44
45 Q = BinHeap()
46 for _ in range(int(input())):
47     operation = list(map(int, input().split()))
48     if operation[0] == 1:
49         Q.heappush(operation[1])
50     else:
51         print(Q.heappop())
52

```

2、AVL树

二叉搜索树中，如果每一个子节点都小于等于根节点，那么会导致查找效率大大降低，这也就是我们使用平衡二叉搜索树的原因。*AVL*树的平衡因子（左子树和右子树高度之差）绝对值小于1，因此可以大大加快查找效率。

```

1 class Node:
2     def __init__(self, val):
3         self.val = val
4         self.left = None
5         self.right = None
6         self.height = 1
7
8 class AVL:
9     def __init__(self):
10         self.root = None
11
12     def insert(self, v):

```

```

13         if self.root is None:
14             self.root = Node(v)
15         else:
16             self.root = self._insert(v, self.root)
17
18     def _insert(self, v, node):
19         if node is None:
20             return Node(v)
21         elif v < node.val:
22             node.left = self._insert(v, node.left)
23         else:
24             node.right = self._insert(v, node.right)
25             node.height = 1 + max(self._get_height(node.left),
self._get_height(node.right))
26         balance = self._get_balance(node)
27         if balance > 1:
28             if v < node.left.val:
29                 return self.rotate_right(node)
30             else:
31                 node.left = self.rotate_left(node.left)
32                 return self.rotate_right(node)
33         elif balance < -1:
34             if v > node.right.val:
35                 return self.rotate_left(node)
36             else:
37                 node.right = self.rotate_right(node.right)
38                 return self.rotate_left(node)
39         return node
40
41     def _get_height(self, node):
42         if node is None:
43             return 0
44         return node.height
45
46     def _get_balance(self, node):
47         if node is None:
48             return 0
49         return self._get_height(node.left) - self._get_height(node.right)
50
51     def rotate_left(self, node):
52         nd = node.right
53         tmp = nd.left
54         nd.left = node
55         node.right = tmp
56         node.height = 1 + max(self._get_height(node.left),
self._get_height(node.right))
57         nd.height = 1 + max(self._get_height(nd.left), self._get_height(nd.right))
58         return nd
59
60     def rotate_right(self, node):
61         nd = node.left
62         tmp = nd.right

```

```

63         nd.right = node
64         node.left = tmp
65         node.height = 1 + max(self._get_height(node.left),
self._get_height(node.right))
66         nd.height = 1 + max(self._get_height(nd.left), self._get_height(nd.right))
67         return nd
68
69     def _prefix(self, node):
70         if node is None:
71             return []
72         return [node.val]+self._prefix(node.left)+self._prefix(node.right)
73
74     def prefix(self):
75         return ' '.join(map(str, self._prefix(self.root)))
76

```

三、题目

1、求二叉树的高度和叶子数目

<http://dsbpython.openjudge.cn/dspythonbook/P0610/>

先写一个`TreeNode`的类，然后用列表进行状态的更新，再找到哪个是根节点，最后就可以进行高度计算和叶子数数了。

```

1  class TreeNode:
2      def __init__(self, val):
3          self.val = val
4          self.left = None
5          self.right = None
6
7  def build(n, nodes):
8      tree = [TreeNode(i) for i in range(n)]
9      check = set()
10     for k in range(n):
11         l, r = nodes[k]
12         if l != -1:
13             tree[k].left = tree[l]
14             check.add(l)
15         if r != -1:
16             tree[k].right = tree[r]
17             check.add(r)
18     for p in range(n):
19         if p not in check:
20             return tree[p]
21
22 def height(root):
23     if root is None:

```

```

24         return 0
25     return max(height(root.left), height(root.right))+1
26
27 def count(root):
28     if root is None:
29         return 0
30     if root.left is None and root.right is None:
31         return 1
32     return count(root.left)+count(root.right)
33
34 n = int(input())
35 nodes = [list(map(int, input().split())) for _ in range(n)]
36 root = build(n, nodes)
37 print(height(root)-1, count(root))
38

```

2、Disk Tree

<http://cs101.openjudge.cn/dsapre/01760/>

虽然没有写类，但是其实已经很好地锻炼了树的思想，怎么建&怎么输出，这都是很有考究的。这里使用的 *setdefault* 方法，很有意思，或许之后还会用到。

```

1  tree = {}
2  for _ in range(int(input())):
3      components = input().split('\\')
4      node = tree
5      for tmp in components:
6          node = node.setdefault(tmp, {})
7  def Print(tree, depth):
8      for key in sorted(tree.keys()):
9          print(' '*depth+key)
10         Print(tree[key], depth+1)
11 Print(tree, 0)
12

```

3、重建二叉树

<http://cs101.openjudge.cn/dsapre/02255/>

最经典的题目，通过两种遍历来得到另一种遍历。

```

1  class TreeNode:
2      def __init__(self, val):
3          self.val = val
4          self.left = None
5          self.right = None
6

```

```

7  def build(prefix, infix):
8      if len(prefix) == 0:
9          return None
10     v = prefix[0]
11     root = TreeNode(v)
12     idx = infix.index(v)
13     root.left = build(prefix[1:idx+1], infix[:idx])
14     root.right = build(prefix[idx+1:], infix[idx+1:])
15     return root
16
17 def postfix(root):
18     if root is None:
19         return ''
20     return postfix(root.left)+postfix(root.right)+root.val
21
22 while True:
23     try:
24         prefix, infix = input().split()
25         print(postfix(build(prefix, infix)))
26     except EOFError:
27         break
28

```

4、四分树

<http://cs101.openjudge.cn/practice/01610/>

这次树的子节点大于二，因此我们用列表来存储。

```

1  class TreeNode:
2      def __init__(self, val):
3          self.val = val
4          self.children = []
5
6  def build(n, matrix):
7      check = sum(sum(row) for row in matrix)
8      if check == 0:
9          return TreeNode('00')
10     elif check == n**2:
11         return TreeNode('01')
12     else:
13         a = [matrix[i][:n//2] for i in range(n//2)]
14         b = [matrix[i][n//2:] for i in range(n//2)]
15         c = [matrix[i][:n//2] for i in range(n//2, n)]
16         d = [matrix[i][n//2:] for i in range(n//2, n)]
17         root = TreeNode('1')
18         for p in [a, b, c, d]:
19             root.children.append(build(n//2, p))
20         return root
21

```



```

22 def get(root):
23     result = ''
24     queue = [root]
25     while queue:
26         node = queue.pop(0)
27         result += node.val
28         queue += node.children
29     return result
30
31 for _ in range(int(input())):
32     n = int(input())
33     matrix = [list(map(int, input().split())) for _ in range(n)]
34     ans = get(build(n, matrix))
35     p = int(ans, 2)
36     print(hex(p)[2:].upper())
37

```

5、表达式·表达式树·表达式求值

<http://cs101.openjudge.cn/practice/05430/>

有一说一，这道题能顺着做下来，感觉树这方面就没啥问题了？首先使用栈进行中序转后序，然后建树，之后需要注意树的打印，最后再进行一个带入求值。

```

1  class TreeNode:
2      def __init__(self, val):
3          self.val = val
4          self.left = None
5          self.right = None
6
7  def trans(infix):
8      stack = []
9      result = []
10     operator = {'+': 1, '-': 1, '*': 2, '/': 2}
11     for token in infix:
12         if token.isalpha():
13             result.append(token)
14         elif token == '(':
15             stack.append(token)
16         elif token == ')':
17             while stack and stack[-1] != '(':
18                 result.append(stack.pop())
19             stack.pop()
20         else:
21             while stack and stack[-1] in operator and operator[token] <=
operator[stack[-1]]:
22                 result.append(stack.pop())
23             stack.append(token)
24     while stack:
25         result.append(stack.pop())

```

```

26     return result
27
28 def build(postfix):
29     stack = []
30     for char in postfix:
31         if char in '+-*/':
32             node = TreeNode(char)
33             node.right = stack.pop()
34             node.left = stack.pop()
35         else:
36             node = TreeNode(char)
37             stack.append(node)
38     return stack[0]
39
40 def depth(root):
41     if root is None:
42         return 0
43     return max(depth(root.left), depth(root.right))+1
44
45 def Print(root, d):
46     if d == 0:
47         return root.val
48     graph = [ ' '*(2**d-1)+root.val+' '*(2**d-1) ]
49     graph.append(' '*(2**d-2)+'/' if root.left is not None else ' ')
50             '+' '+(\'\\\' if root.right is not None else ' ')+ ' '*(2**d-2))
51     d -= 1
52     l = Print(root.left, d) if root.left is not None else [ ' '*(2**d-1) ]*(2**d-1)
53     r = Print(root.right, d) if root.right is not None else [ ' '*(2**d-1) ]*(
(2**d-1)
54     for i in range(2**d-1):
55         graph.append(l[i]+' '+r[i])
56     return graph
57
58 def cal(root):
59     if root.val.isalpha():
60         return my_dict[root.val]
61     else:
62         lv = cal(root.left)
63         rv = cal(root.right)
64         return int(eval(str(lv)+root.val+str(rv)))
65
66 infix = list(input())
67 postfix = trans(infix)
68 root = build(postfix)
69 my_dict = {}
70 for _ in range(int(input())):
71     a, b = input().split()
72     my_dict[a] = b
73 print(''.join(postfix))
74 d = depth(root)
75 result = Print(root, d-1)
76 for part in result:

```

```

77     print(part)
78     print(cal(root))
79

```

6、树的转换

<http://cs101.openjudge.cn/practice/04081/>

这个题不难，但是很好地诠释了什么叫做“左儿子右兄弟”，当时我理解了半天还没明白的东西hhh。

```

1  s = list(input())
2  a, b, h, nh = 0, 0, 0, 0
3  now = [0]
4  for char in s:
5      if char == 'd':
6          a, b = a+1, b+1
7          now.append(a)
8      else:
9          a, b = now.pop(), b-1
10     nh = max(nh, a)
11     h = max(h, b)
12     print(f'{h} => {nh}')
13

```

7、森林的带度数层次序列存储

<http://dsbpython.openjudge.cn/dspythonbook/P0770/>

通过层次遍历的结果进行建树。

```

1  from collections import deque
2
3  class TreeNode:
4      def __init__(self, val, degree):
5          self.val = val
6          self.degree = degree
7          self.children = []
8
9  def build(nodes):
10     queue = deque()
11     a, b = nodes.pop(0)
12     root = TreeNode(a, b)
13     f = root
14     queue.append(f)
15     while queue:
16         node = queue.popleft()
17         for _ in range(node.degree):
18             a, b = nodes.pop(0)

```

```

19         tmp = TreeNode(a, b)
20         node.children.append(tmp)
21         queue.append(tmp)
22     return root
23
24 def postfix(root):
25     ans = []
26     for node in root.children:
27         ans += postfix(node)
28     ans.append(root.val)
29     return ans
30
31 result = []
32 for _ in range(int(input())):
33     lst = list(input().split())
34     n = len(lst)//2
35     nodes = [[lst[2*i], int(lst[2*i+1])] for i in range(n)]
36     result += postfix(build(nodes))
37 print(' '.join(result))
38

```

8、二叉搜索树的层次遍历

<http://dsbpython.openjudge.cn/dspythonbook/P1320/>

二叉搜索树，就是左子节点小于根节点，根节点小于右子节点。知道这个剩下的就好做了。

```

1 class TreeNode:
2     def __init__(self, val):
3         self.val = val
4         self.left = None
5         self.right = None
6
7 def build(root, k):
8     if root is None:
9         return TreeNode(k)
10    if root.val < k:
11        root.right = build(root.right, k)
12    if root.val > k:
13        root.left = build(root.left, k)
14    return root
15
16 def operate(root):
17     result = []
18     queue = [root]
19     while queue:
20         node = queue.pop(0)
21         result.append(node.val)
22         if node.left is not None:
23             queue.append(node.left)

```

```

24         if node.right is not None:
25             queue.append(node.right)
26         return ' '.join(map(str, result))
27
28 nodes = list(map(int, input().split()))
29 root = None
30 for node in nodes:
31     root = build(root, node)
32 print(operate(root))
33

```

9、括号嵌套二叉树

<http://dsbpython.openjudge.cn/dspythonbook/P0680/>

使用栈对括号和逗号进行处理，之后进行遍历即可。

```

1  class TreeNode:
2      def __init__(self, val):
3          self.val = val
4          self.children = []
5
6  def build(nodes):
7      if not nodes:
8          return None
9      if '(' not in nodes:
10         return TreeNode(nodes[0])
11     root = TreeNode(nodes[0])
12     stack = []
13     tmp = []
14     tree = []
15     for char in nodes[2: -1]:
16         if char == '(':
17             stack.append(char)
18         elif char == ')':
19             stack.pop()
20         elif char == ',' and not stack:
21             tree.append(tmp)
22             tmp = []
23             continue
24         tmp.append(char)
25     tree.append(tmp)
26     for t in tree:
27         root.children.append(build(t))
28     return root
29
30 def prefix(root):
31     if root is None:
32         return ''
33     ans = root.val

```

```

34     for child in root.children:
35         ans += prefix(child)
36     return ans
37
38 def postfix(root):
39     if root is None:
40         return ''
41     ans = ''
42     for child in root.children:
43         ans += postfix(child)
44     ans += root.val
45     return ans
46
47 nodes = list(input())
48 root = build(nodes)
49 print(prefix(root))
50 print(postfix(root))
51

```

10、哈夫曼编码树

<http://cs101.openjudge.cn/practice/22161/>

非常经典的一种算法！需要注意，我们之前的建树过程都是自上向下建立（先建立根节点，之后不断添加子树），这里我们自下向上，逐渐向上，“拼凑”出一棵完整的树。

```

1  import heapq
2
3  class TreeNode:
4      def __init__(self, weight, val=None):
5          self.weight = weight
6          self.val = val
7          self.left = None
8          self.right = None
9      def __lt__(self, other):
10         if self.weight == other.weight:
11             return self.val < other.val
12         return self.weight < other.weight
13
14 def build(nodes):
15     Q = []
16     for p, char in nodes:
17         heapq.heappush(Q, TreeNode(p, char))
18     while len(Q) > 1:
19         l = heapq.heappop(Q)
20         r = heapq.heappop(Q)
21         merge = TreeNode(l.weight + r.weight)
22         merge.left = l
23         merge.right = r
24         heapq.heappush(Q, merge)

```

```

25     return Q[0]
26
27 def encode(root):
28     codes = {}
29     def operate(node, code):
30         if node.val is not None:
31             codes[node.val] = code
32         else:
33             operate(node.left, code+'0')
34             operate(node.right, code+'1')
35     operate(root, '')
36     return codes
37
38 def encoding(codes, s):
39     ans = ''
40     for p in s:
41         ans += codes[p]
42     return ans
43
44 def decoding(root, s):
45     ans = ''
46     node = root
47     for p in s:
48         if p == '0':
49             node = node.left
50         else:
51             node = node.right
52         if node.val is not None:
53             ans += node.val
54             node = root
55     return ans
56
57 nodes = []
58 for _ in range(int(input())):
59     a, b = input().split()
60     nodes.append([int(b), a])
61 root = build(nodes)
62 codes = encode(root)
63 while True:
64     try:
65         s = list(input())
66         if s[0].isdigit():
67             print(decoding(root, s))
68         else:
69             print(encoding(codes, s))
70     except EOFError:
71         break
72

```

11、01 Tree

<https://www.luogu.com.cn/problem/CF1919D>

为了不让大家看英文，所以选的洛谷上面的这个（好吧是我不想看.....）。感觉这道题也是需要从下向上对树进行处理的思维，挨个对叶子节点进行处理（因为输入数据是按照顺序来的），逐渐向上，相当于逐渐收到最终的根节点，如果根节点符合要求，那就输出YES。当然了，处理的途中加入一些特判也是需要的。

```
1  for _ in range(int(input())):
2      n = int(input())
3      stack = [[-1, 0]]
4      check = 0
5      top = 1
6      judge = True
7      for x in map(int, input().split()):
8          x += 1
9          check += (x == 1)
10         if check > 1:
11             judge = False
12             break
13         if x <= stack[-1][1]:
14             judge = False
15             break
16         elif x == stack[-1][1]+1:
17             if stack[-1][0] == stack[-1][1]:
18                 stack.pop()
19             else:
20                 stack[-1][1] -= 1
21         elif top <= x-2:
22             stack.append([top, x-2])
23             top = x
24         if stack != [[-1, -1]] or check != 1 or not judge:
25             print('NO')
26         else:
27             print('YES')
```

12、Preorder

<https://www.luogu.com.cn/problem/CF1671E>

这道题我没有使用dp（对于dp我是一直用的不熟练，树形dp会在后面出现的），直接递归，取子树的先序遍历，进行比较，一旦有不一样的就乘以2。需要注意，在返回先序遍历时应当比较左子树和右子树的字典序，这相当于进行了判重。

```
1  n = int(input())
2  s = '' + list(input())
3  ans, mod = 1, 998244353
4  def recursion(x):
5      global ans
6      if x >= 1 << (n-1):
```



```

7         return s[x]
8     l = recursion(x*2)
9     r = recursion(x*2+1)
10    if l != r:
11        ans = (ans*2) % mod
12    if l > r:
13        l, r = r, l
14    return s[x]+l+r
15 recursion(1)
16 print(ans)
17

```

13、文件结构“图”

<http://cs101.openjudge.cn/practice/02775/>

这道题和 *Disk Tree* 比较像，都属于目录树的内容（当时找到 *Disk Tree* 的时候就记得有一道也很有意思，但是没找到。。。），对于树的思想，是很有帮助的。

```

1  op = '|      '
2
3  def dfs(n, k):
4      file = []
5      c = n
6      while True:
7          tmp = input()
8          if tmp[0] == '#':
9              return
10         if c == 1:
11             print(f'DATA SET {k}:')
12             print('ROOT')
13             c += 1
14         if tmp[0] == '*':
15             file.sort()
16             for f in file:
17                 print(op*(n-1)+f)
18             print()
19             dfs(1, k+1)
20             return
21         elif tmp[0] == 'd':
22             print(op*n+tmp)
23             dfs(n+1, k)
24         elif tmp[0] == ']':
25             file.sort()
26             for f in file:
27                 print(op*(n-1)+f)
28             return
29         else:
30             file.append(tmp)
31

```

```
32 dfs(1, 1)
33
```

14、魔族密码

<https://www.luogu.com.cn/problem/P1481>

这道题就是可以用dp或字典树来解决，这里都展示出来，都是很有用的。

dp:

```
1 n = int(input().strip())
2 s = [input().strip() for _ in range(n)]
3 dp = [1 for _ in range(n)]
4 for i in range(1, n):
5     for j in range(i-1, -1, -1):
6         if s[i].find(s[j]) == 0:
7             dp[i] = max(dp[j]+1, dp[i])
8 print(max(dp))
9
```

字典树:

```
1 tree = [[0 for _ in range(26)] for _ in range(2*10**5)]
2 word = [0 for _ in range(10**6)]
3 n = int(input().strip())
4 tot, ans = 0, 0
5
6 def insert(s):
7     global tot, ans
8     u, res = 0, 0
9     for c in s:
10        k = ord(c)-ord('a')
11        if tree[u][k] == 0:
12            tot += 1
13            tree[u][k] = tot
14            u = tree[u][k]
15            res += word[u]
16        word[u] += 1
17        ans = max(ans, res+1)
18
19 for _ in range(n):
20     insert(input().strip())
21 print(ans)
22
```

15、没有上司的舞会

这道题是非常经典的一道树形dp，感觉其实有了dp的思想后，主要需要的就是应用在各种各样的数据结构上面。

```
1 import sys
2 sys.setrecursionlimit(1 << 30)
3
4 n = int(input())
5 r = [0]+[int(input()) for _ in range(n)]
6 dp = [[0, r[i]] for i in range(n+1)]
7 graph = [[] for _ in range(n+1)]
8 check = set()
9 for _ in range(n-1):
10     u, v = map(int, input().split())
11     graph[v].append(u)
12     check.add(u)
13 boss = [i for i in range(1, n+1) if i not in check][0]
14
15 def dfs(x):
16     for y in graph[x]:
17         dfs(y)
18         dp[x][0] += max(dp[y][0], dp[y][1])
19         dp[x][1] += dp[y][0]
20
21 dfs(boss)
22 print(max(dp[boss]))
23
```

Part 5: 图

最后一个部分！

一、定义

图论(*Graph theory*)是数学的一个分支，图是图论的主要研究对象。图(*Graph*)是由若干给定的顶点及连接两顶点的边所构成的图形，这种图形通常用来描述某些事物之间的某种特定关系。顶点用于代表事物，连接两顶点的边则用于表示两个事物间具有这种关系。

图(*Graph*)是一个二元组 $G = (V(G), E(G))$ 。其中 $V(G)$ 是非空集，称为点集(*vertex set*)，对于 V 中的每个元素，我们称其为顶点(*vertex*)或节点(*node*)，简称点； $E(G)$ 为 $V(G)$ 各节点之间边的集合，称为边集(*edge set*)。

图有多种，包括无向图(*undirected graph*)，有向图(*directed graph*)，混合图(*mixed graph*)等。

若 G 为无向图，则 E 中的每个元素为一个无序二元组 (u, v) ，称作无向边(*undirected edge*)，简称为边(*edge*)，其中 $u, v \in V$ 。设 $e = (u, v)$ ，则 u 和 v 称为 e 的端点(*endpoint*)。

若 G 为有向图，则 E 中的每个元素为一个有序二元组 (u, v) ，有时也写作 $u \rightarrow v$ ，称作**有向边** (*directed edge*)或**弧**(*arc*)，在不引起混淆的情况下也可以称作**边**(*edge*)。设 $e = u \rightarrow v$ ，则此时 u 称为 e 的**起点**(*tail*)， v 称为 e 的**终点**(*head*)，起点和终点也称为 e 的**端点**(*endpoint*)，并称 u 是 v 的直接前驱， v 是 u 的直接后继。

若 G 为混合图，则 E 中既有**有向边**，又有**无向边**。

若 G 的每条边 $e_k = (u_k, v_k)$ 都被赋予一个数作为该边的**权**，则称 G 为**赋权图**。如果这些权都是正实数，就称 G 为**正权图**。

与一个顶点 v 关联的边的条数称作该顶点的**度**(*degree*)，记作 $d(v)$ 。特别地，对于边 (v, v) ，则每条这样的边要对 $d(v)$ 产生2的贡献。

握手定理（图论基本定理）：对于任何无向图 $G = (V, E)$ ，有 $\sum_{v \in V} d(v) = 2|E|$ 。

对一张图，所有节点的度数的最小值称为 G 的**最小度**(*minimum degree*)，记作 $\delta(G)$ ；最大值称为**最大度**(*maximum degree*)，记作 $\Delta(G)$ 。即： $\delta(G) = \min_{v \in G} d(v)$ ， $\Delta(G) = \max_{v \in G} d(v)$ 。

在有向图 $G = (V, E)$ 中，以一个顶点 v 为起点的边的条数称为该节点的**出度**(*out-degree*)，记作 $d^+(v)$ 。以一个顶点 v 为终点的边的条数称为该节点的**入度**(*in-degree*)，记作 $d^-(v)$ 。显然 $d^+(v) + d^-(v) = d(v)$ 。

图 G 的点数 $|V(G)|$ 也被称作图 G 的**阶**(*order*)。

形象地说，图是由若干点以及连接点与点的边构成的。

二、图的表示方法（图的存储）

1、直接存边

方法：

使用一个数组来存边，数组中的每个元素都包含一条边的起点与终点（带边权的图还包含边权）。（或者使用多个数组分别存起点，终点和边权）

参考代码：

```
1 class Edge:
2     def __init__(self, u=0, v=0):
3         self.u = u
4         self.v = v
5
6 n, m = map(int, input().split())
7 graph = [Edge() for _ in range(m)]
8 vis = [False for _ in range(n)]
9 for i in range(m):
10     graph[i].u, graph[i].v = map(int, input().split())
11
12 def find_edge(a, b):
13     for k in range(m):
14         if graph[k].u == a and graph[k].v == b:
```

```

15         return True
16     return False
17
18 def dfs(p):
19     if vis[p]:
20         return
21     vis[p] = True
22     for j in range(m):
23         if graph[j].u == p:
24             dfs(graph[j].v)
25

```

复杂度：

查询是否存在某条边： $O(m)$ 。

遍历一个点的所有出边： $O(m)$ 。

遍历整张图： $O(nm)$ 。

空间复杂度： $O(m)$ 。

应用：

由于直接存边的遍历效率低下，一般不用于遍历图。

在 *Kruskal* 算法中，由于需要将边按边权排序，需要直接存边。

在有的题目中，需要多次建图（如建一遍原图，建一遍反图），此时既可以使用多个其它数据结构来同时存储多张图，也可以将边直接存下来，需要重新建图时利用直接存下的边来建图。

2、邻接矩阵

方法：

使用一个二维数组 *graph* 来存边，其中 *graph*[*u*][*v*] 为 1 表示存在 *u* 到 *v* 的边，为 0 表示不存在。如果是带边权的图，可以在 *graph*[*u*][*v*] 中存储 *u* 到 *v* 的边的边权。

参考代码：

```

1  n, m = map(int, input().split())
2  graph = [[False for _ in range(n+1)] for _ in range(n+1)]
3  vis = [False for _ in range(n+1)]
4  for _ in range(m):
5      u, v = map(int, input().split())
6      graph[u][v] = True
7
8  def find_edge(a, b):
9      return graph[a][b]
10
11 def dfs(p):
12     if vis[p]:
13         return

```

```

14     vis[p] = True
15     for q in range(n+1):
16         if graph[p][q]:
17             dfs(q)
18

```

复杂度：

查询是否存在某条边： $O(1)$ 。

遍历一个点的所有出边： $O(n)$ 。

遍历整张图： $O(n^2)$ 。

空间复杂度： $O(n^2)$ 。

应用：

邻接矩阵只适用于没有重边（或重边可以忽略）的情况。

其最显著的优点是可以 $O(1)$ 查询一条边是否存在。

由于邻接矩阵在稀疏图上效率很低（尤其是在点数较多的图上，空间无法承受），所以一般只会在稠密图上使用邻接矩阵。

3、邻接表

方法：

为了实现稀疏连接的图，更高效的方式是使用邻接表。在邻接表实现中，我们为图对象的所有顶点保存一个主列表，同时为每一个顶点对象都维护一个列表，其中记录了与它相连的顶点。

参考代码：

```

1  n, m = map(int, input().split())
2  graph = {i: [] for i in range(1, n+1)}
3  for _ in range(m):
4      u, v = map(int, input().split())
5      graph[u].append(v)
6  vis = [False for _ in range(n+1)]
7
8  def find_edge(a, b):
9      for tmp in graph[a]:
10         if tmp == b:
11             return True
12     return False
13
14  def dfs(p):
15      if vis[p]:
16         return
17      vis[p] = True
18      for q in graph[p]:
19         dfs(q)

```

复杂度：

查询是否存在 u 到 v 的边： $O(d^+(v))$ 。（若排序，二分查找可以降低复杂度）

遍历点 u 的所有出边： $O(d^+(v))$ 。

遍历整张图： $O(n + m)$ 。

空间复杂度： $O(m)$ 。

应用：

存各种图都很适合，除非有特殊需求（如需要快速查询一条边是否存在，且点数较少，可以使用邻接矩阵）。

尤其适用于需要对一个点的所有出边进行排序的场合。

4、链式前向星

如果说邻接表是不好写但效率高，邻接矩阵是好写但效率低的话，前向星就是一个相对中庸的数据结构。前向星固然好写，但效率并不高。而在优化为链式前向星后，效率也得到了较大的提升。虽然说，世界上对链式前向星的使用并不是很广泛，但在不愿意写复杂的邻接表的情况下，链式前向星也是一个很优秀的数据结构。

方法：

本质就是用链表实现的邻接表，但邻接表存的是点，而链式前向星存的是边。

参考代码：

```

1  class Edge:
2      def __init__(self, to, w, nxt):
3          self.to = to
4          self.w = w
5          self.next = nxt
6
7  n, m = map(int, input().split())
8  edge = [Edge(0, 0, 0) for _ in range(m)]
9  head = [-1 for _ in range(n+1)]
10 cnt = 0
11 for _ in range(m):
12     u, v, w = map(int, input().split())
13     edge[cnt].to = v
14     edge[cnt].w = w
15     edge[cnt].next = head[u]
16     head[u] = cnt
17     cnt += 1
18 vis = [False for _ in range(n+1)]
19
20 def find_edge(a, b):
21     i = head[a]
22     while i != -1:
23         if edge[i].to == b:

```

```

24         return True
25         i = edge[i].next
26     return False
27
28 def dfs(p):
29     if vis[p]:
30         return
31     vis[p] = True
32     i = head[u]
33     while i != -1:
34         dfs(edge[i].to)
35         i = edge[i].next
36

```

复杂度：

查询是否存在 u 到 v 的边： $O(d^+(u))$ 。

遍历点 u 的所有出边： $O(d^+(u))$ 。

遍历整张图： $O(n + m)$ 。

空间复杂度： $O(m)$ 。

应用：

存各种图都很适合，但不能快速查询一条边是否存在，也不能方便地对一个点的出边进行排序。

优点是边是带编号的，有时会非常有用，而且如果 cnt 的初始值为奇数，存双向边时 $i \oplus 1$ 即是 i 的反边（常用于网络流）。

三、图的遍历

I. 广度优先搜索

1、词梯

<http://cs101.openjudge.cn/practice/28046/>

这道题显然是需要使用bfs的，但是如果仅仅是使用邻接表储存这个图然后遍历的话，会出现MLE的情况。因此，我们使用通配符匹配，既然字符长度只有四个，那么我们考虑使用"xx_x"、"x_xx"这样的形式储存，同时进行遍历，能够减小内存并保证时间不会增加太多。

```

1  from collections import deque
2  def check(a, b):
3      for k in range(len(a)):
4          if a[k] == '_':
5              continue
6          if a[k] != b[k]:
7              return False

```



```

8         return True
9
10    n = int(input())
11    graph = {}
12    degree = {}
13    vis = {}
14    for _ in range(n):
15        word = input()
16        vis[word] = False
17        for p in range(4):
18            tmp = word[:p]+'_'+word[p+1:]
19            if word[:p]+'_'+word[p+1:] not in graph:
20                graph[tmp] = [word]
21                degree[tmp] = 1
22            else:
23                graph[tmp].append(word)
24                degree[tmp] += 1
25
26    def bfs():
27        start, ending = input().split()
28        queue = deque()
29        queue.append([start, [start]])
30        vis[start] = True
31        for p in range(4):
32            tmp = start[:p] + '_' + start[p + 1:]
33            degree[tmp] -= 1
34        while queue:
35            wd, now = queue.popleft()
36            if wd == ending:
37                print(*now)
38                return
39            for p in range(4):
40                tmp = wd[:p] + '_' + wd[p + 1:]
41                if degree[tmp] > 0:
42                    for wor in graph[tmp]:
43                        if not vis[wor]:
44                            vis[wor] = True
45                            degree[tmp] -= 1
46                            queue.append([wor, now + [wor]])
47        print('NO')
48    bfs()
49

```

2、子串变换

<https://www.luogu.com.cn/problem/P1032>

挺简单的一道搜索题，需要注意的是直接使用 *find* 函数的话，只能找到第一个字符串，后面若有重复的话就找不到。这种情况很好解决，同时记得开 *vis* 集合用来判重。

```

1  from collections import deque
2
3  A, B = input().split()
4  graph = {}
5  while True:
6      try:
7          a, b = input().split()
8          if a in graph:
9              graph[a].append(b)
10         else:
11             graph[a] = [b]
12     except EOFError:
13         break
14
15  queue = deque()
16  queue.append([A, 0])
17  vis = set()
18  vis.add(A)
19  while queue:
20      now, t = queue.popleft()
21      if t > 10:
22          continue
23      if now == B:
24          print(t)
25          break
26      for key in graph:
27          tmp = []
28          st = 0
29          while True:
30              idx = now[st:].find(key)
31              if idx == -1:
32                  break
33              tmp.append(idx+st)
34              st = idx+st+len(key)
35          for idx in tmp:
36              ni = idx+len(key)
37              for v in graph[key]:
38                  new = now[:idx]+v+now[ni:]
39                  if new not in vis:
40                      vis.add(new)
41                      queue.append([now[:idx]+v+now[ni:], t+1])
42  else:
43      print('NO ANSWER!')
44

```

3、树的计数

<https://www.luogu.com.cn/problem/P1232>

先定序，这个喜欢用哪种确定都可以；然后判断哪个点必须分，哪个点不可能分，哪个点可分不可分（这种无论分不分都对后续状态无影响），分别加1、0、0.5即可；最后输出即可。

```
1 n = int(input())
2 sd, sb = [0 for _ in range(n)], [0 for _ in range(n)]
3 dfn = list(map(int, input().split()))
4 bfn = list(map(int, input().split()))
5 for i in range(n):
6     sd[dfn[i]-1] = i
7     sb[bfn[i]-1] = i
8 for j in range(n):
9     dfn[j] = sb[dfn[j]-1]
10    bfn[j] = sd[bfn[j]-1]
11 d = [0 for _ in range(n)]
12 d[0], d[1] = 1, -1
13 ans = 2
14 for i in range(n-1):
15     if bfn[i] > bfn[i+1]:
16         d[i] += 1
17         d[i+1] -= 1
18         ans += 1
19 for i in range(n-1):
20     if dfn[i]+1 < dfn[i+1]:
21         d[dfn[i]] += 1
22         d[dfn[i+1]] -= 1
23 w = 0
24 for i in range(n-1):
25     w += d[i]
26     if not w:
27         ans += 0.5
28 print(f'{ans:.3f}')
29
```

4、Artificial Idiot

<http://cs101.openjudge.cn/practice/28052/>

这道题就是使用heap+记忆化搜索即可，主要感觉题目情景以及描述比较新颖。同时数据比较逆天.....

```
1 import heapq
2
3 dir = [(-1, -1), (-1, 0), (0, -1), (0, 1), (1, 0), (1, 1)]
4 n = int(input())
5 tmp = [list(map(int, input().split())) for _ in range(n)]
6 memo = [[float('inf') for _ in range(n)] for _ in range(n)]
7
8 a, b = sum(tmp[i].count(1) for i in range(n)), sum(tmp[j].count(2) for j in range(n))
9 matrix = [[0 for _ in range(n)] for _ in range(n)]
10 if a == b:
11     for i in range(n):
```

```

12         for j in range(n):
13             if tmp[i][j] == 1:
14                 matrix[j][i] = 1
15             elif tmp[i][j] == 2:
16                 matrix[j][i] = -1
17     elif a > b:
18         for i in range(n):
19             for j in range(n):
20                 if tmp[i][j] == 1:
21                     matrix[i][j] = -1
22                 elif tmp[i][j] == 2:
23                     matrix[i][j] = 1
24     else:
25         print(-1)
26         exit()
27
28     queue = []
29     for i in range(n):
30         if matrix[i][0] == 1:
31             heapq.heappush(queue, (0, i, 0))
32             memo[i][0] = 0
33         elif matrix[i][0] == 0:
34             heapq.heappush(queue, (1, i, 0))
35             memo[i][0] = 1
36     while queue:
37         t, x, y = heapq.heappop(queue)
38         if y == n-1:
39             print(t)
40             exit()
41         for a, b in dir:
42             nx, ny = x+a, y+b
43             if 0 <= nx < n and 0 <= ny < n:
44                 if matrix[nx][ny] == -1:
45                     continue
46                 elif matrix[nx][ny] == 1:
47                     if t < memo[nx][ny]:
48                         memo[nx][ny] = t
49                         heapq.heappush(queue, (t, nx, ny))
50                 else:
51                     if t+1 < memo[nx][ny]:
52                         memo[nx][ny] = t+1
53                         heapq.heappush(queue, (t+1, nx, ny))
54     print(-1)
55

```

5、最后的迷宫

<https://www.luogu.com.cn/problem/P2199>

这道题耗了好多时间.....本来以为是个挺普通的bfs，结果反复TLE、MLE，只能说这道题的数据确实会出现比较极端的情况，所以也可以比较好地要求对于存储结构的利用，以及对数据的预先处理。看题解，好像可以压成一维处理，应该是行的，降低内存，但是脑子笨就直接写二维了（也就导致了如果稍微哪里写的过头就会超时或者超内存.....）。

```
1  from collections import deque
2
3  dir = [(-1, 0), (0, -1), (0, 1), (1, 0), (-1, -1), (-1, 1), (1, -1), (1, 1)]
4  n, m = map(int, input().split())
5  maze = [list(input()) for _ in range(n)]
6  vis = [[0]*m for _ in range(n)]
7
8  def bfs(x0, y0):
9      queue = deque()
10     queue.append((x0, y0))
11     if vis[x0][y0] == -1:
12         return 0
13     vis[x0][y0] = 1
14     while queue:
15         x, y = queue.popleft()
16         for k in range(4):
17             nx, ny = x+dir[k][0], y+dir[k][1]
18             if 0 <= nx < n and 0 <= ny < m and maze[nx][ny] == 'O':
19                 if vis[nx][ny] == -1:
20                     return vis[x][y]
21                 if vis[nx][ny] == 0:
22                     queue.append((nx, ny))
23                     vis[nx][ny] = vis[x][y]+1
24     return 'Poor Harry'
25
26 def change(p):
27     p.clear()
28     for _ in range(n):
29         p.append([0]*m)
30
31 def mark(p, q):
32     for a, b in dir:
33         r, c = p, q
34         while 0 <= r < n and 0 <= c < m and maze[r][c] == 'O':
35             vis[r][c] = -1
36             r += a
37             c += b
38
39 while True:
40     d, j, w, v = map(int, input().split())
41     if d == j == w == v == 0:
42         break
43     change(vis)
44     mark(d-1, j-1)
45     print(bfs(w-1, v-1))
46
```

6、孤舟蓑笠翁

<https://www.luogu.com.cn/problem/P4730>

II. 深度优先搜索

1、正方形破坏者

<http://cs101.openjudge.cn/practice/01084/>

使用估价函数剪枝的一道题目，建议去看这篇博客：https://blog.csdn.net/2301_79402523/article/details/137194237，也是我写的。

```
1  import copy
2  import sys
3  sys.setrecursionlimit(1 << 30)
4  found = False
5
6  def check1(x, tmp):
7      for y in graph[x]:
8          if tmp[y]:
9              return False
10     return True
11
12 def check2(x):
13     for y in graph[x]:
14         if judge[y]:
15             return False
16     return True
17
18 def estimate():
19     cnt = 0
20     tmp = copy.deepcopy(judge)
21     for x in range(1, total+1):
22         if check1(x, tmp):
23             cnt += 1
24             for u in graph[x]:
25                 tmp[u] = True
26     return cnt
27
28 def dfs(t):
29     global found
30     if t + estimate() > limit:
31         return
32     for x in range(1, total+1):
33         if check2(x):
34             for y in graph[x]:
35                 judge[y] = True
36                 dfs(t+1)
```

```

37         judge[y] = False
38         if found:
39             return
40     return
41     found = True
42
43 for _ in range(int(input())):
44     n = int(input())
45     lst = list(map(int, input().split()))
46     d, m, nums, total = 2*n+1, lst[0], lst[1:], 0
47     graph = {}
48     for i in range(n):
49         for j in range(n):
50             for k in range(1, n+1):
51                 if i+k <= n and j+k <= n:
52                     total += 1
53                     graph[total] = []
54                     for p in range(1, k+1):
55                         graph[total] += [d*i+j+p, d*(i+p)+j-n, d*(i+p)+j-n+k, d*
(i+k)+j+p]
56     judge = [False for _ in range(2*n*(n+1)+1)]
57     for num in nums:
58         judge[num] = True
59     limit = estimate()
60     found = False
61     while True:
62         dfs(0)
63         if found:
64             print(limit)
65             break
66     limit += 1
67

```

2、骑士周游

<http://cs101.openjudge.cn/practice/28050/>

这道题我们姑且不考虑一些奇技淫巧，正常用dfs做的话，是会超时的，所以我们需要使用*Warnsdorff*算法，每一次选择的时候都选择可行子节点最少的子节点，从而达到最快得到答案的目的。

```

1  import sys
2  sys.setrecursionlimit(1 << 30)
3
4  dir = [(-2, -1), (-2, 1), (-1, -2), (-1, 2), (1, -2), (1, 2), (2, -1), (2, 1)]
5  n = int(input())
6  s, e = map(int, input().split())
7  edge = {}
8  vis = {}
9  degree = {}
10 for i in range(n):

```

```

11     for j in range(n):
12         vis[(i, j)] = False
13         edge[(i, j)] = []
14         for a, b in dir:
15             ni, nj = i+a, j+b
16             if 0 <= ni < n and 0 <= nj < n:
17                 edge[(i, j)].append((ni, nj))
18         degree[(i, j)] = len(edge[(i, j)])
19
20 def dfs(x, y):
21     if all(vis[key] for key in vis):
22         print('success')
23         exit()
24     if degree[(x, y)] <= 0:
25         return
26     for p, q in dir:
27         nx, ny = x+p, y+q
28         if 0 <= nx < n and 0 <= ny < n and not vis[(nx, ny)]:
29             degree[(nx, ny)] -= 1
30     edge[(x, y)].sort(key=lambda m: degree[m])
31     for r, c in edge[(x, y)]:
32         if not vis[(r, c)]:
33             vis[(r, c)] = True
34             dfs(r, c)
35             vis[(r, c)] = False
36     for p, q in dir:
37         nx, ny = x+p, y+q
38         if 0 <= nx < n and 0 <= ny < n and not vis[(nx, ny)]:
39             degree[(nx, ny)] += 1
40
41 vis[(s, e)] = True
42 dfs(s, e)
43 print('fail')
44

```

3、The Rotation Game

<http://cs101.openjudge.cn/practice/02286/>

又一道估价函数剪枝的，不得不说这种做法真的可以大大减少时间复杂度。试图总结一下，dfs中的剪枝可以分成以下的类别：

- 最优性剪枝（发现已经次优于当前最优，那么就可以放弃后续搜索）
- 可行性剪枝（判断是否违反了问题的约束条件）
- 启发式剪枝（提前预测哪些分支很可能不会产生最优解，从而加速搜索）
- 对称性剪枝（利用对称性剪枝）
- 记忆化剪枝（记录先前搜索过的状态进行判重）


```

1 line = {'A': [0, 2, 6, 11, 15, 20, 22],
2         'B': [1, 3, 8, 12, 17, 21, 23],
3         'C': [10, 9, 8, 7, 6, 5, 4],
4         'D': [19, 18, 17, 16, 15, 14, 13],
5         'E': [23, 21, 17, 12, 8, 3, 1],
6         'F': [22, 20, 15, 11, 6, 2, 0],
7         'G': [13, 14, 15, 16, 17, 18, 19],
8         'H': [4, 5, 6, 7, 8, 9, 10]
9     }
10 center = [6, 7, 8, 11, 12, 15, 16, 17]
11
12 def check():
13     for i in range(8):
14         if mp[center[i]] != mp[center[0]]:
15             return False
16     return True
17
18 def move(r):
19     tmp = [mp[line[r][i]] for i in range(7)]
20     for j in range(7):
21         mp[line[r][j-1]] = tmp[j]
22
23 def move_back(c):
24     tmp = [mp[line[c][i]] for i in range(7)]
25     for j in range(-1, 6):
26         mp[line[c][j+1]] = tmp[j]
27
28 def diff(t):
29     cnt = 0
30     for i in range(8):
31         if mp[center[i]] != t:
32             cnt += 1
33     return cnt
34
35 def h():
36     return min(diff(1), diff(2), diff(3))
37
38 def dfs(dep, max_d):
39     if check():
40         print(''.join(ans))
41         return True
42     if dep+h() > max_d:
43         return False
44     for letter in 'ABCDEFGH':
45         ans.append(letter)
46         move(letter)
47         if dfs(dep+1, max_d):
48             return True
49         ans.pop()
50         move_back(letter)
51     return False
52

```

```

53 while True:
54     mp = list(map(int, input().split()))
55     if mp == [0]:
56         break
57     ans = []
58     if check():
59         print('No moves needed')
60     else:
61         limit = 1
62         while True:
63             if dfs(0, limit):
64                 break
65             limit += 1
66     print(mp[6])
67

```

4、魔法指纹

<https://www.luogu.com.cn/problem/P1822>

这是一道挺好的dfs题目，很好地考察了与队列的结合。思路是很简单的，从7开始不断扩展并进行判断即可。

```

1  from collections import deque
2  a = int(input())
3  b = int(input())
4  queue = deque([7])
5  ans = 1 if a <= 7 <= b else 0
6
7  def dfs(x, y, t):
8      global ans
9      if y > b:
10         return
11     if x == 0:
12         last = y // (t // 10)
13         if not last:
14             return
15         dfs(x, y+last*t, t*10)
16         if a <= y <= b:
17             ans += 1
18         if t < b:
19             queue.append(y)
20         return
21     last, nxt = y // (t // 10), x % 10
22     x //= 10
23     if last-nxt >= 0:
24         dfs(x, y+(last-nxt)*t, t*10)
25     if nxt and last+nxt < 10:
26         dfs(x, y+(last+nxt)*t, t*10)
27
28 while queue:

```

```

29     x = queue.popleft()
30     for i in range(10):
31         dfs(x, i, 10)
32 print(ans)
33

```

5、切蛋糕

<https://www.luogu.com.cn/problem/P1528>

没想到.....是我拷贝的时候写多了，导致时间复杂度太大。思路是清晰的，贪心+dfs+二分，主要是剪枝不算好想，不过也还好，倒是真没想到自己会在拷贝这么一个小坑里面栽个大跟头。

```

1  import sys
2  sys.setrecursionlimit(1 << 30)
3
4  def dfs(x, st, cake):
5      global w
6      if x == 0:
7          return True
8      if total-w < prev[mid]:
9          return False
10     for j in range(st, n+1):
11         if cake[j] >= mouth[x]:
12             cake[j] -= mouth[x]
13             if cake[j] < mouth[1]:
14                 w += cake[j]
15             if mouth[x] == mouth[x-1]:
16                 if dfs(x-1, j, cake):
17                     return True
18             else:
19                 if dfs(x-1, 1, cake):
20                     return True
21             if cake[j] < mouth[1]:
22                 w -= cake[j]
23             cake[j] += mouth[x]
24     return False
25
26 n = int(input())
27 cake = sorted([0]+[int(input()) for _ in range(n)])
28 total = sum(cake)
29 m = int(input())
30 mouth = sorted([0]+[int(input()) for _ in range(m)])
31 prev = [0 for _ in range(m+1)]
32 for i in range(1, m+1):
33     prev[i] = prev[i-1]+mouth[i]
34 left, right = 1, m
35 while mouth[right] > cake[-1] and right > 0:
36     right -= 1
37 ans = 0

```

```

38 while left <= right:
39     w = 0
40     mid = (left+right)//2
41     if dfs(mid, 1, cake[:]):
42         ans = mid
43         left = mid+1
44     else:
45         right = mid-1
46 print(ans)
47

```

6、兔兔与蛋蛋游戏

<https://www.luogu.com.cn/problem/P1971>

四、拓扑排序

1、Sorting It All Out

<http://cs101.openjudge.cn/practice/01094/>

其实就是正常的拓扑排序，只是每次添加新的边时都需要进行判断。

```

1 def topo_sort(x):
2     if vis[x] == -1:
3         return -1
4     if pos[x] != -1:
5         return pos[x]
6     vis[x] = -1
7     p = n
8     for i in range(len(graph[x])):
9         p = min(p, topo_sort(graph[x][i]))
10        if p == -1:
11            return -1
12        topo[p-1] = x
13        pos[x], vis[x] = p-1, 0
14        return p-1
15
16 while True:
17     n, m = map(int, input().split())
18     if n == m == 0:
19         break
20     graph = {i: [] for i in range(n)}
21     lst = [input() for _ in range(m)]
22     topo = [0 for _ in range(n)]
23     ans = n
24     for i in range(m):

```

```

25     graph[ord(lst[i][0])-ord('A')].append(ord(lst[i][2])-ord('A'))
26     vis = [0 for _ in range(n)]
27     pos = [-1 for _ in range(n)]
28     for j in range(n):
29         ans = min(ans, topo_sort(j))
30     if ans == -1:
31         print(f'Inconsistency found after {i+1} relations.')
32         break
33     elif ans == 0:
34         print(f'Sorted sequence determined after {i+1} '
35               f'relations: {"".join([chr(topo[k]+ord("A")) for k in
range(n)])}.')
36         break
37     if ans > 0:
38         print('Sorted sequence cannot be determined.')
39

```

2、神经网络

<https://www.luogu.com.cn/problem/P1038>

使用拓扑排序，最后进行节点的判断并输出即可。

```

1  from collections import deque
2  n, p = map(int, input().split())
3  nerve = {i: 0 for i in range(1, n+1)}
4  queue = deque()
5  for k in range(1, n+1):
6      c, u = map(int, input().split())
7      nerve[k] = c
8      if c == 0:
9          nerve[k] -= u
10     else:
11         queue.append(k)
12  in_degree = {i: 0 for i in range(1, n+1)}
13  out_degree = {i: 0 for i in range(1, n+1)}
14  W = {i: {} for i in range(1, n+1)}
15  for _ in range(p):
16      i, j, w = map(int, input().split())
17      W[i][j] = w
18      in_degree[j] += 1
19      out_degree[i] += 1
20  while queue:
21      node = queue.popleft()
22      for j in W[node]:
23          nerve[j] += W[node][j]*nerve[node]
24          in_degree[j] -= 1
25          if in_degree[j] == 0 and nerve[j] > 0:
26              queue.append(j)
27  flag = True

```

```

28 for q in range(1, n+1):
29     if out_degree[q] == 0:
30         if nerve[q] > 0:
31             flag = False
32             print(q, nerve[q])
33 if flag:
34     print('NULL')
35

```

3、Elaxia的路线

<https://www.luogu.com.cn/problem/P2149>

这道题很不错，首先需要跑几次*dijkstra*（两对点，也就是四个点，分别出发跑一次）来获取*dis*数组，随后做拓扑排序，得到最短路的*DAG*，最后使用队列做判断，找到最长链即可。总体来说，考察的内容很丰富，对于时间和内存的限制没有那么死但也没有很宽松，可以很好地考察对于图论相关算法知识以及对于数据结构的理解应用。

```

1  from collections import deque
2  import heapq
3  import sys
4  input = sys.stdin.readline
5  def dijkstra(x, k):
6      dis[k][x] = 0
7      queue = []
8      heapq.heappush(queue, (0, x))
9      while queue:
10         t, p = heapq.heappop(queue)
11         if t != dis[k][p]:
12             continue
13         for q in graph[p]:
14             if dis[k][p]+graph[p][q] < dis[k][q]:
15                 dis[k][q] = dis[k][p]+graph[p][q]
16                 heapq.heappush(queue, (dis[k][p]+graph[p][q], q))
17
18  n, m = map(int, input().split())
19  x1, y1, x2, y2 = map(int, input().split())
20  graph = {i: {} for i in range(1, n+1)}
21  for _ in range(m):
22      u, v, w = map(int, input().split())
23      graph[u][v] = graph[v][u] = w
24  dis = [[float('inf') for _ in range(n+1)] for _ in range(4)]
25  dijkstra(x1, 0)
26  dijkstra(y1, 1)
27  dijkstra(x2, 2)
28  dijkstra(y2, 3)
29
30  in_degree = [0 for _ in range(n+1)]
31  for u in range(1, n+1):
32      for v in list(graph[u]):
33          if dis[0][u]+graph[u][v]+dis[1][v] == dis[0][y1]:

```

```

34         in_degree[v] += 1
35     else:
36         del graph[u][v]
37
38 ans = 0
39 f, g = [0 for _ in range(n+1)], [0 for _ in range(n+1)]
40 queue = deque()
41 queue.append(x1)
42 while queue:
43     u = queue.popleft()
44     ans = max(ans, f[u], g[u])
45     for v in graph[u]:
46         in_degree[v] -= 1
47         if dis[2][u]+graph[u][v]+dis[3][v] == dis[2][y2]:
48             f[v] = max(f[v], f[u]+graph[u][v])
49         if dis[3][u]+graph[u][v]+dis[2][v] == dis[2][y2]:
50             g[v] = max(g[v], g[u]+graph[u][v])
51         if in_degree[v] == 0:
52             queue.append(v)
53 print(ans)
54

```

4、道路

<https://www.luogu.com.cn/problem/P2505>

这道题.....应该用python没有办法通过它的全部测试点了（C++跑都需要六七百毫秒），这里跑n次dijkstra，每次跑完之后取拓扑序然后反着来统计两个数组（一个记录起点到某一点的最短路数，一个记录终点到某一点的最短路数），二者相乘再累加即可。

```

1  import heapq, sys
2  input = sys.stdin.readline
3
4  n, m = map(int, input().split())
5  G = [[] for _ in range(n+1)]
6  for i in range(m):
7      x, y, z = map(int, input().split())
8      G[x].append([y, z, i])
9  ans, cnt = [0]*m, [0]*(n+1)
10 mod = 10**9+7
11
12 def dijkstra(x0):
13     tmp, dis = [0]*(n+1), [float('inf')]*(n+1)
14     queue = []
15     tmp[x0] = 1
16     dis[x0] = 0
17     heapq.heappush(queue, (0, x0))
18     dag = []
19     while queue:
20         d, u = heapq.heappop(queue)

```

```

21         if d != dis[u]:
22             continue
23         dag.append(u)
24         for p in G[u]:
25             v, w, idx = p
26             if dis[u]+w < dis[v]:
27                 dis[v] = dis[u]+w
28                 tmp[v] = tmp[u]
29                 heapq.heappush(queue, (dis[v], v))
30             elif dis[u]+w == dis[v]:
31                 tmp[v] += tmp[u]
32     dag.reverse()
33     for u in dag:
34         cnt[u] = 1
35         for f in G[u]:
36             v, w, idx = f
37             if dis[u]+w == dis[v]:
38                 cnt[u] = (cnt[u]+cnt[v]) % mod
39                 ans[idx] = (ans[idx]+tmp[u]*cnt[v]%mod) % mod
40
41 for i in range(1, n+1):
42     dijkstra(i)
43
44 for j in range(m):
45     print(ans[j])
46

```

5、摄像头

<https://www.luogu.com.cn/problem/P2712>

一道很水的拓扑排序，算是拿去练练手感吧。数据范围要是写错会RE，同时需要注意摄像头不是连续的。

```

1  from collections import deque
2  graph = {i: [] for i in range(1, 1001)}
3  n = int(input())
4  now = []
5  in_degree = {i: 0 for i in range(1, 1001)}
6  for _ in range(n):
7      tmp = list(map(int, input().split()))
8      now.append(tmp[0])
9      for p in tmp[2:]:
10         graph[tmp[0]].append(p)
11         in_degree[p] += 1
12  queue = deque()
13  for w in now:
14     if in_degree[w] == 0:
15         queue.append(w)
16  ans = n
17  while queue:

```



```

18     u = queue.popleft()
19     ans -= 1
20     for v in graph[u]:
21         in_degree[v] -= 1
22         if in_degree[v] == 0 and v in now:
23             queue.append(v)
24 if ans == 0:
25     print('YES')
26 else:
27     print(ans)
28

```

五、生成树

1、买礼物

<https://www.luogu.com.cn/problem/P1194>

这里使用了 *Kruskal* 算法，简单说就是我们找边，只要找到 $n - 1$ 条最小边，同时每次向生成树添加边的时候判断会不会出现连接之前已经连起来的节点（判断的时候使用并查集）。

```

1  def find(x):
2      if p[x] != x:
3          p[x] = find(p[x])
4      return p[x]
5
6  def union(x, y):
7      fx = find(x)
8      fy = find(y)
9      if fx != fy:
10         p[fx] = fy
11         return True
12     return False
13
14  a, b = map(int, input().split())
15  edges = [(0, i, a) for i in range(1, b)]
16  for i in range(b):
17      tmp = list(map(int, input().split()))
18      for j in range(i):
19          if tmp[j] != 0:
20              edges.append((i, j, tmp[j]))
21  p = [i for i in range(b)]
22  edges.sort(key=lambda x: x[2])
23  ans = a
24  for u, v, w in edges:
25      if union(u, v):
26          ans += w
27  print(ans)

```

2、Arctic Network

<http://dsbpython.openjudge.cn/dspythonbook/P1240/>

同样是 *Kruskal* 算法，这里不需要连全部边， $p - s$ 条即可。类似的题目，洛谷也有一道：口袋的天空，<https://www.luogu.com.cn/problem/P1195>。

```

1  import math
2
3  def cal(a, b):
4      return math.sqrt(pow(a[0]-b[0], 2)+pow(a[1]-b[1], 2))
5
6  def find(x):
7      if f[x] != x:
8          f[x] = find(f[x])
9      return f[x]
10
11 def union(x, y):
12     fx = find(x)
13     fy = find(y)
14     if fx != fy:
15         f[fx] = fy
16         return True
17     return False
18
19 for _ in range(int(input())):
20     s, p = map(int, input().split())
21     outposts = [list(map(int, input().split())) for _ in range(p)]
22     f = [i for i in range(p)]
23     edges = []
24     for j in range(p):
25         for i in range(j):
26             edges.append((i, j, cal(outposts[i], outposts[j])))
27     edges.sort(key=lambda x: x[2])
28     ans, cnt = 0, 0
29     for u, v, w in edges:
30         if union(u, v):
31             ans = max(ans, w)
32             cnt += 1
33             if cnt >= p-s:
34                 break
35     print(f'{ans:.2f}')
36

```

3、Cheering up the Cow G

<https://www.luogu.com.cn/problem/P2916>

这道题还是使用 *Kruskal* 算法 (*Prim* 在 OJ 上面大家做了很多, 而在洛谷上面找到的比较合适的题目用 python 又都过不了), 主要对于权值的处理比较有趣。

```
1 def find(x):
2     if f[x] != x:
3         f[x] = find(f[x])
4     return f[x]
5
6 def union(x, y):
7     fx, fy = find(x), find(y)
8     if fx != fy:
9         f[fx] = fy
10    return True
11    return False
12
13 n, p = map(int, input().split())
14 cows = [0]+[int(input()) for _ in range(n)]
15 f = [i for i in range(n+1)]
16 edges = []
17 for _ in range(p):
18     u, v, w = map(int, input().split())
19     edges.append((u, v, cows[u]+cows[v]+2*w))
20 edges.sort(key=lambda x: x[2])
21 ans = 0
22 for u, v, w in edges:
23     if union(u, v):
24         ans += w
25 print(ans+min(cows[1:]))
26
```

4、瞎破坏

<http://dsbpython.openjudge.cn/dspythonbook/P1290/>

这道题最开始没理解意思.....首先我们先想想, 最小生成树有几条边? 而有环的图又至少有几条边, 由此我们就发现, 我们只需要去掉一颗最小生成树就可以了。

```
1 def find(x):
2     if f[x] != x:
3         f[x] = find(f[x])
4     return f[x]
5
6 def union(x, y):
7     fx, fy = find(x), find(y)
8     if fx != fy:
9         f[fx] = fy
10    return True
11    return False
```

```

12
13 n, m = map(int, input().split())
14 f = [i for i in range(n)]
15 edges = []
16 ans = 0
17 for _ in range(m):
18     x, y, v = map(int, input().split())
19     edges.append((x, y, v))
20     ans += v
21 edges.sort(key=lambda x: x[2])
22 cnt = 0
23 for u, v, w in edges:
24     if union(u, v):
25         ans -= w
26         cnt += 1
27         if cnt == n-1:
28             break
29 print(ans)
30

```

5、小店购物

<https://www.luogu.com.cn/problem/P2792>

omg.....这道题卡在手头卡了两三天，终于AC了。首先我们看题面就注意到，这并不是最小生成树，准确说不是无向图的最小生成树，而应该是有向图的最小树形图（和最小生成树一个意思，不过由于有向，所以叫做树形图）。这样的问题有两种算法，一个是朱刘算法，另一个是Tarjan的DMST算法，后者对我来说太难了😭。这里介绍一下朱刘算法。

其实朱刘算法的想法很简单，就是找到n-1条最小边组成树，问题在对于环怎么处理。实际上我们把环看成一个单元进行处理（做个比喻就是当成黑盒子），就可以了，这也是朱刘算法的精髓。oiwiki上是进行这样的总结的——

- 1、对于每个点，选择指向它的边权最小的那条边。
- 2、如果没有环，算法终止；否则进行缩环并更新其他点到环的距离。

emmm感觉用文字可能说的不太清楚，这里放一道模板题以及代码+注释叭~

P4716 【模板】最小树形图(<https://www.luogu.com.cn/problem/P4716>)

```

1 class Edge:
2     def __init__(self, u, v, w):
3         self.u = u
4         self.v = v
5         self.w = w
6
7 def zhu_liu(n, root):
8     ans = 0
9     while True:
10         pre, vis, in_vector = [-1]*n, [-1]*n, [float('inf')]*n
11         # 寻找并记录每个点的最小入边

```

```

12     in_vector[root] = 0
13     for i in range(m):
14         if edges[i].u != edges[i].v and edges[i].w < in_vector[edges[i].v]:
15             pre[edges[i].v] = edges[i].u
16             in_vector[edges[i].v] = edges[i].w
17     # 有孤立点，不存在最小树形图
18     for i in range(n):
19         if i != root and in_vector[i] == float('inf'):
20             return -1
21     # 找有向环
22     loop = 0 # 记录环的个数，用来在之后缩环成点提供编号
23     circle = [-1]*n
24     for i in range(n):
25         ans += in_vector[i]
26         v = i
27         # 向前遍历环，中止情况如下：
28         # 1、出现带有相同标记的点，成环（这个情况是我们之后需要做处理的）
29         # 2、节点属于其他环，说明进入了其他环
30         # 3、遍历到根节点了
31         while vis[v] != i and circle[v] == -1 and v != root:
32             vis[v] = i
33             v = pre[v]
34         # 成环才能进入while循环，把环内的点用circle进行标记（同样是为之后的缩环成点做准备）
35         if v != root and circle[v] == -1:
36             while circle[v] != loop:
37                 circle[v] = loop
38                 v = pre[v]
39             loop += 1
40         # 如果缩到没有环了，那么就找全了
41         if loop == 0:
42             break
43         # 否则将所有孤立点也当作自环看待
44         for i in range(n):
45             if circle[i] == -1:
46                 circle[i] = loop
47                 loop += 1
48         # 统一地进行缩环
49         for j in range(m):
50             v = edges[j].v
51             edges[j].u = circle[edges[j].u]
52             edges[j].v = circle[edges[j].v]
53             # 如果边不属于同一个环，则进行更新
54             if edges[j].u != edges[j].v:
55                 edges[j].w -= in_vector[v]
56         n = loop
57         root = circle[root]
58     return ans
59
60 n, m, r = map(int, input().split())
61 edges = []
62 for i in range(m):
63     u, v, w = map(int, input().split())

```

```

64     edges.append(Edge(u-1, v-1, w))
65     print(zhu_liu(n, r-1))
66

```

然后我们来看这道题，首先读题发现，应该要进行编号的重整，不买的商品不加入图中，然后我们不难想到，只要添加一个超级源点，就可以跑朱刘算法了，所以用原价与源点相连，接着贪心地想到，对于需要买 n ($n > 1$)个的商品，我们可以先当成1个来买，最后加上 $(n - 1)$ 乘以其最低价格就可以。思路想清楚之后套朱刘算法的模板就能够轻松AC~

```

1  class Edge:
2      def __init__(self, u, v, w):
3          self.u = u
4          self.v = v
5          self.w = w
6
7  def zhu_liu(n, m, root, edges):
8      ans = 0
9      while True:
10         pre, vis, in_vector = [-1]*n, [-1]*n, [float('inf')]*n
11         in_vector[root] = 0
12         for i in range(m):
13             if edges[i].u != edges[i].v and edges[i].w < in_vector[edges[i].v]:
14                 pre[edges[i].v] = edges[i].u
15                 in_vector[edges[i].v] = edges[i].w
16         for i in range(n):
17             if i != root and in_vector[i] == float('inf'):
18                 return -1
19         loop = 0
20         circle = [-1]*n
21         for i in range(n):
22             ans += in_vector[i]
23             v = i
24             while vis[v] != i and circle[v] == -1 and v != root:
25                 vis[v] = i
26                 v = pre[v]
27             if v != root and circle[v] == -1:
28                 while circle[v] != loop:
29                     circle[v] = loop
30                     v = pre[v]
31                 loop += 1
32         if loop == 0:
33             break
34         for i in range(n):
35             if circle[i] == -1:
36                 circle[i] = loop
37                 loop += 1
38         for j in range(m):
39             v = edges[j].v
40             edges[j].u = circle[edges[j].u]

```

```

41         edges[j].v = circle[edges[j].v]
42         if edges[j].u != edges[j].v:
43             edges[j].w -= in_vector[v]
44     n = loop
45     root = circle[root]
46     return ans
47
48 n = int(input())
49 P, nums, edges = [], [], []
50 my_dict = {}
51 for i in range(1, n+1):
52     tmp = input().split()
53     c, q = float(tmp[0]), int(tmp[1])
54     if q:
55         P.append(c)
56         nums.append(q)
57         my_dict[i] = len(P)
58         edges.append(Edge(0, len(P), c))
59 m = int(input())
60 for _ in range(m):
61     tmp = input().split()
62     u, v, w = int(tmp[0]), int(tmp[1]), float(tmp[2])
63     if u in my_dict and v in my_dict:
64         edges.append(Edge(my_dict[u], my_dict[v], w))
65         P[my_dict[v]-1] = min(P[my_dict[v]-1], w)
66 result = zhu_liu(len(P)+1, len(edges), 0, edges)
67 for i in range(len(P)):
68     if nums[i] > 1:
69         result += (nums[i]-1)*P[i]
70 print(f'{result:.2f}')
71

```

Part 6: 结语

终于写完了.....突然有点儿自我感动了，写一点结语吧。

该总结于二月末开始写，从线性表、栈、队列开始，中间穿插内排序算法，最重要的树、图两个部分放在最后。

感谢闫老师、助教和全体选择这门课的同学，在做不出来题目的时候，闫老师提供的数据和解法总会有很大帮助，而大家在群聊里面的讨论，真的让我受益匪浅，选这门课之前以为自己对于这些数据结构、算法已经了解颇多，交流之后才发现原来只知皮毛，差得远，够的学。而我也在之后的互相交流中，学习到了很多新鲜而深刻的知识，非常感谢大家！

这篇总结中，知识点偏少（我认为大部分知识点闫老师课上都有讲，而且网上也很好搜），更多的是选择了一些我觉得比较有意思或有代表性的题目放在不同的部分里面，选题来自POJ或洛谷（其中有一些题目在洛谷上python实在过不了），希望能够给大家一些启发。

说实话，这样的总结做起来虽然很累，但做的过程中对于我的编程水平很有提升，而且做完之后很有成就感。这有可能是我选择的最后一门与信科相关的课程，非常高兴能够以用这样的方式做一个收尾，衷心感谢大家的帮助！也希望大家能够在未来的学习（无论什么课程）中，越来越好~