今天我们来制作**poker** 这个牌类游戏

1. understanding（理解问题）

> Start with a vague understanding that you refine into a problem.

2. specify （明确问题）

> Specify how this problem can be made amenable to being coded.

3. design （设计程序）

> Coding

游戏规则：

- 所有可能的手牌

```
• 0- High Card
• 1- One Pair
• 2- Two Pair
• 3- Three of a Kind
• 4- Straight
• 5- Flush
• 6- Full House
• 7- Four of a Kind
• 8- Straight Flush
```

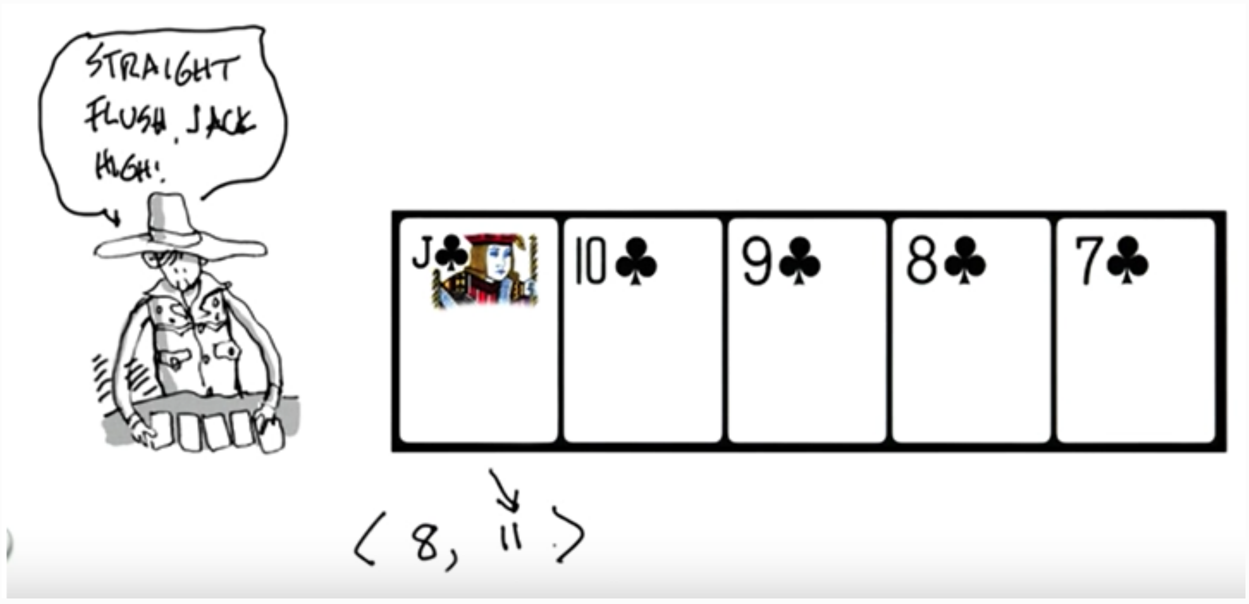
【注：我们用其对应的数字作为得分】

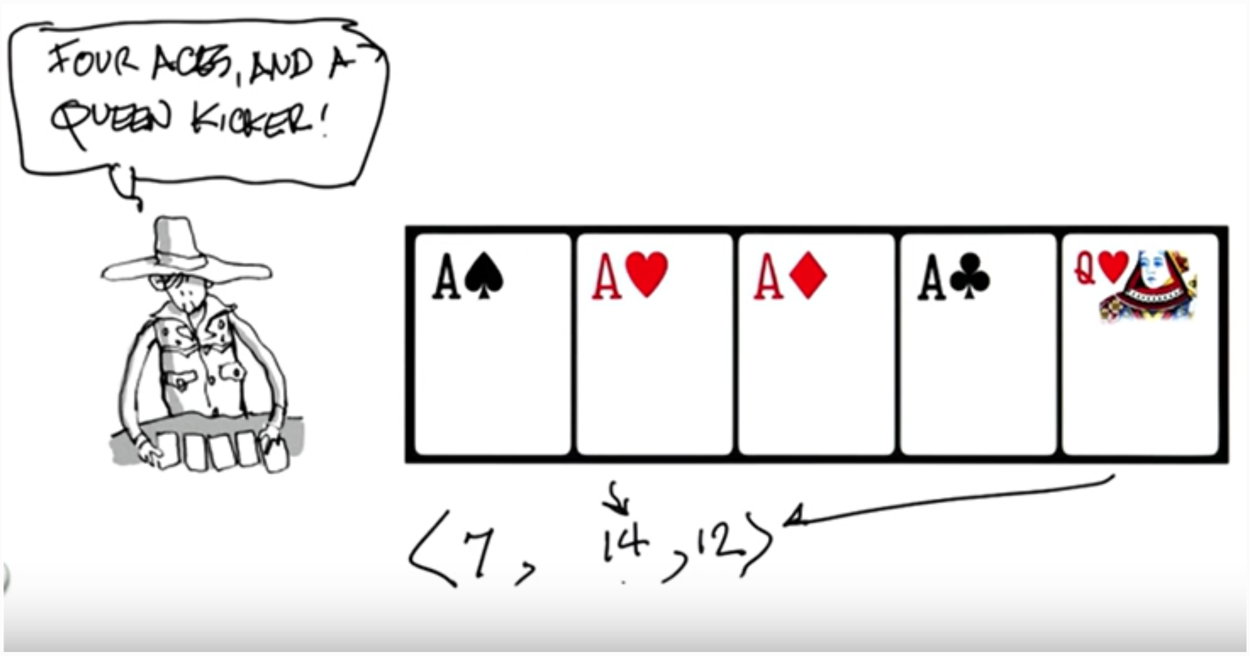
详细游戏规则自行上维基百科上查看

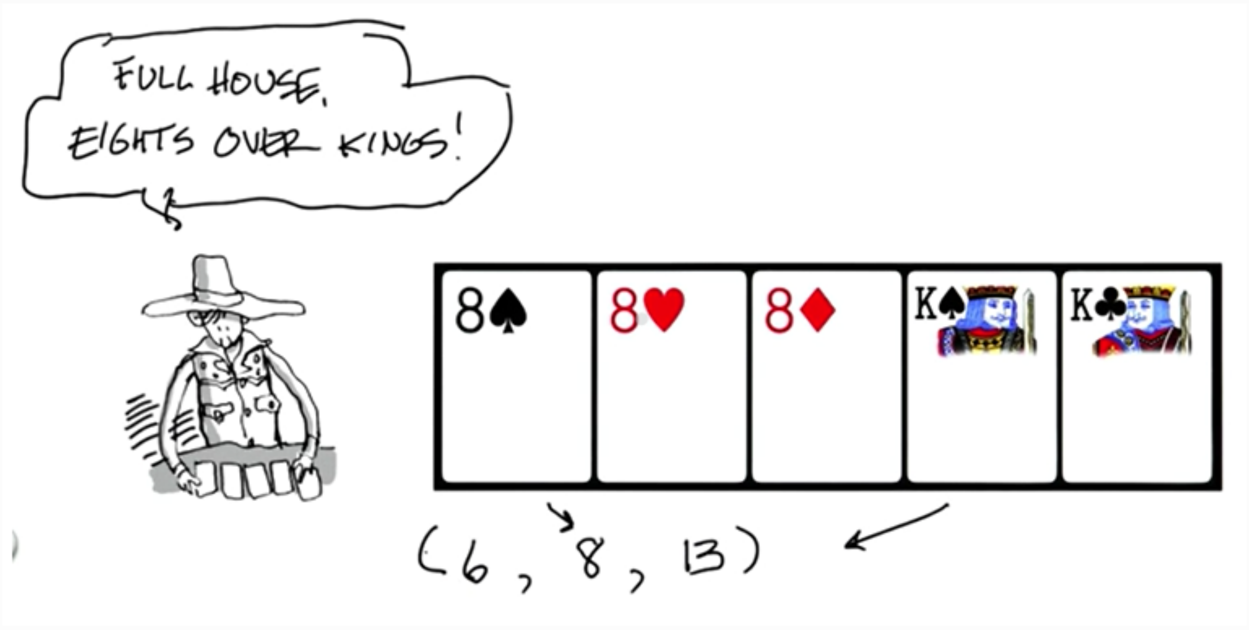
如何在hands 中存储我们的手牌，手牌由数字和花色组成（可以采用如下方式）：

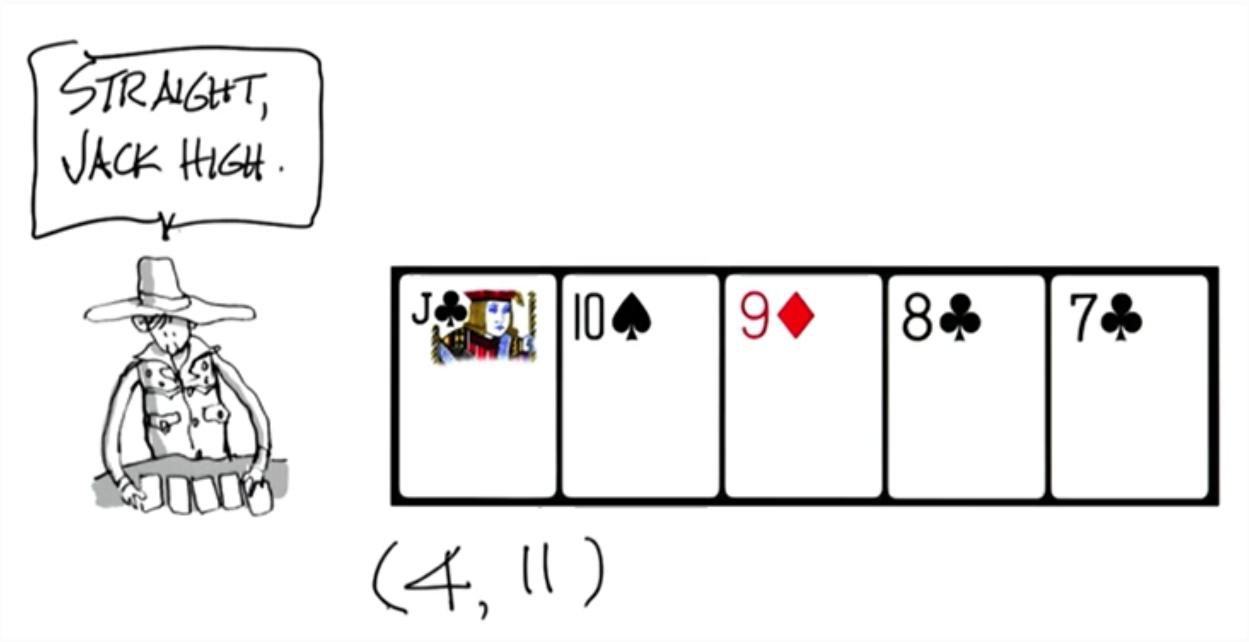
```
sf = ['6C', '7C', '8C', '9C', 'TC'] # Straight Flush

fk = ['9D', '9H', '9S', '9C' ,'7D'] # Four of a Kind

fh = ['TD', 'TC', 'TH', '7C', '7D']  #Full House
```

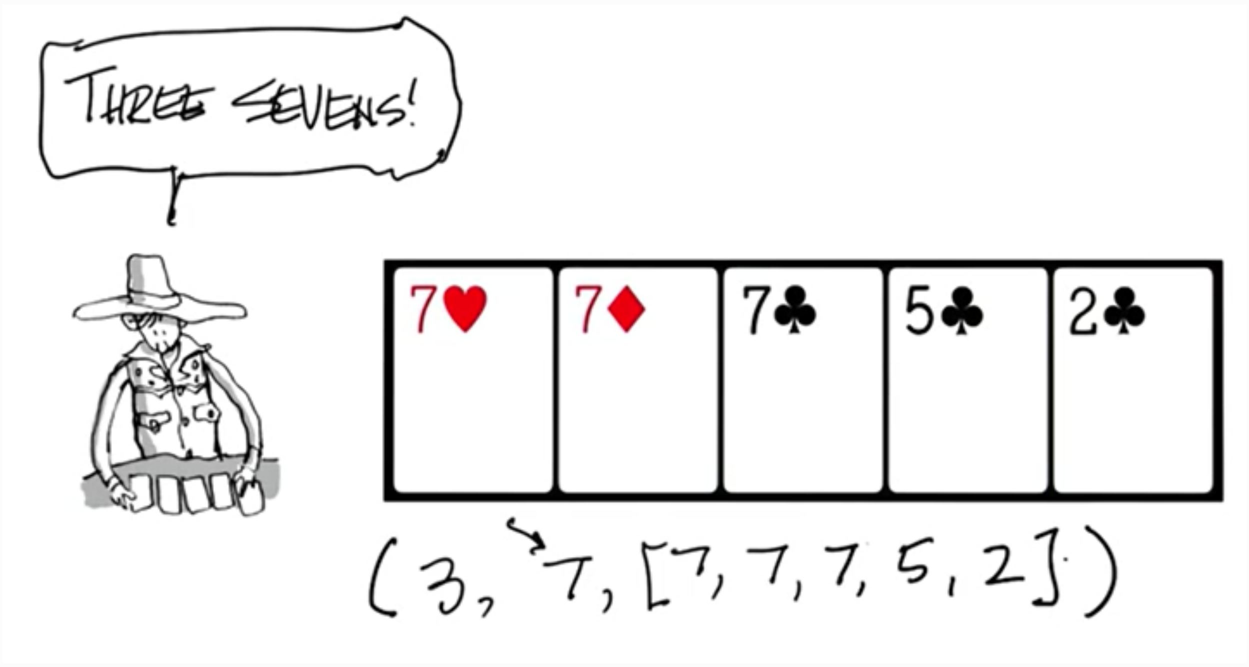
用何种数据结构来存放这些手牌类型：

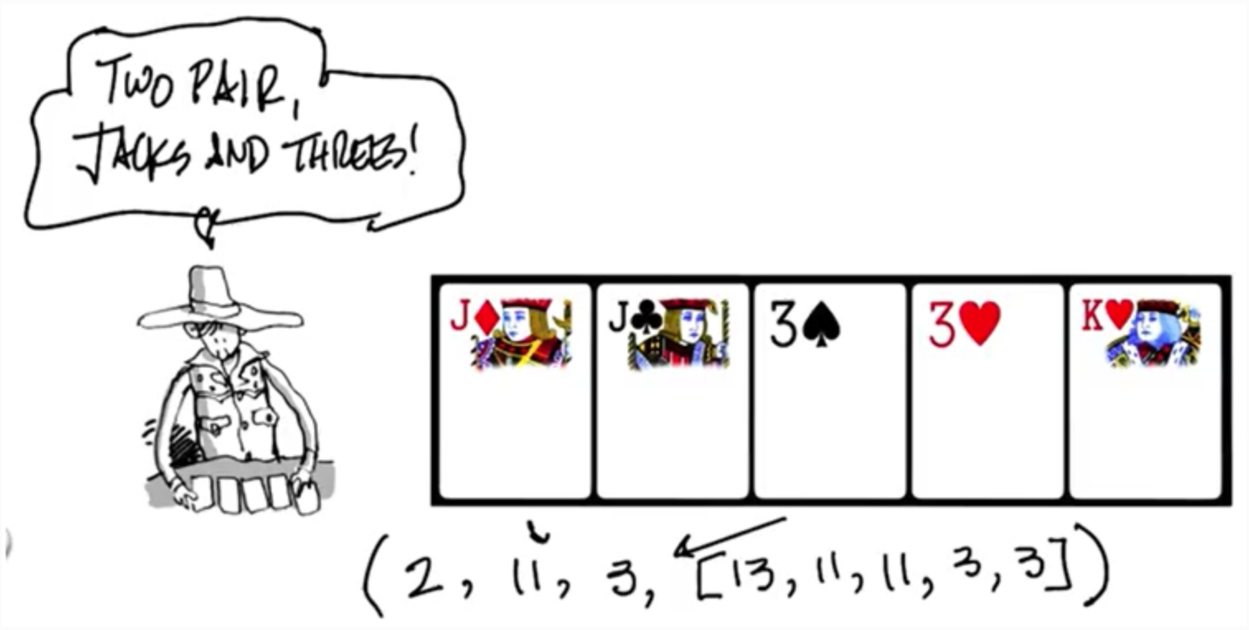





FULL HOUSE, EIGHTS OVER KINGS!
8♠ 8♥ 8♦ K♠ K♣
(6, 8, 13)


STRAIGHT, JACK HIGH.
J♣ 10♠ 9♦ 8♣ 7♣
(4, 11)


THREE SEVENS!
7♥
7♦
7♣
5♣
2♣
(3, 7, [7, 7, 7, 5, 2])


TWO PAIR, JACKS AND THREES!
J♦
J♣
3♠
3♥
K♥
(2, 11, 3, [13, 11, 11, 3, 3])

我们最终要判断哪副手牌可以获胜，所以我们需要一个判断胜负的函数

```
Def poker(hands):
    "return the best hand"
```

进一步分析我们要判断大小，那么我们需要对手牌有一个大小的衡量机制

```
Def hand_rank(hand):
    "return  the rank of hand"
```

有了这个大小的度量机制之后

> return max(hands, key=hand_rank) 就能返回 best hand

于是我们得到我们的poker 函数

```
def poker(hands):
    "Return the best hand: poker([hand,...]) => hand"
    return max(hands, key=hand_rank)
```

接下来我们要完成hand_rank() 函数，因为hand_rank 函数涉及到手牌的类型，所以我们要完成对手牌类型的判断

Straight

> straight(ranks):

```
                  returns True if the hand is a straight.
```

Flush

> flush(hand):

```
                 returns True if the hand is a flush
```

Kind

> kind(n, ranks):

```
                  returns the first rank that the hand has exactly n of.
For A hand with 4 sevens  this function would return 7.
```

two_pair

> two_pair(ranks):

```
                 if there is a two pair, this function returns their
corresponding ranks as a tuple. For example, a hand with 2 twos and 2
fours would cause this function to return (4, 2).
```

card_ranks

> card_ranks(hand):

```
                  returns an ORDERED tuple of the ranks  in a hand
(where the order goes from highest to lowest rank).
```

假定以上函数已经完成，那么我们的hand_rank 函数如下

```
    def hand_rank(hand):
        ranks = card_ranks(hand)
        if straight(ranks) and flush(hand):            # straight
flush
            return (8, max(ranks))
        elif kind(4, ranks):                           # 4 of a kind
            return (7, kind(4, ranks), kind(1, ranks))
        elif kind(3, ranks) and kind(2, ranks):        # full house
            return (6,kind(3, ranks),kind(2, ranks))
        elif flush(hand):                              # flush
            return (5,ranks)
        elif straight(ranks):                          # straight
            return (4,max(ranks))
        elif kind(3, ranks):                           # 3 of a kind
            return (3,kind(3,ranks),ranks)
        elif two_pair(ranks):                          # 2 pair
            return (2,two_pair(ranks),ranks)
        elif kind(2, ranks):                           # kind
            return (1,kind(2,ranks),ranks)
        else:                                          # high card
            return (0,ranks)
```

为了检验我们的程序是否成功，我们常常需要做测试，例如可以通过assert 来检验，样例代码如下

```
    def test():
        "Test cases for the functions in poker program"
        sf = "6C 7C 8C 9C TC".split() # Straight Flush
        fk = "9D 9H 9S 9C 7D".split() # Four of a Kind
        fh = "TD TC TH 7C 7D".split() # Full House
        assert poker([sf, fk, fh]) == sf
        assert poker([fk, fh]) == fk
        assert poker([fh, fh]) == fh
        assert poker([sf]) == sf
        assert poker([sf] + 99*[fh]) == sf
        assert hand_rank(sf) == (8, 10)
        assert hand_rank(fk) == (7, 9, 7)
        assert hand_rank(fh) == (6, 10, 7)
        return 'tests pass'
```

接下来我们来具体实现判断手牌类型的函数

Straight

```
def straight(ranks):
    "Return True if the ordered ranks form a 5-card straight."
    min_num = ranks[0]
    List = [min_num - i for i in range(len(ranks))]
    if List == ranks:
        return True
    else:
        return False
```

或

```
def straight(ranks):
    "return true if the ordered ranks from a 5-card straight"
    return (max(ranks)-min(ranks) == 4) and len(set(ranks)) == 5
```

Flush

```
def flush(hand):
    "Return True if all the cards have the same suit."
    suit = [s for n,s in hand]
    if [suit[0]]*len(suit) == suit:
        return True
    else:
        return False
```

或

```
def flush(hand):
    "return true if all the cards have the same suit"
    suits = [s for r,s in hand]

return len(set(suits)) ==1
```

Kind

```
def kind(n, ranks):
    """Return the first rank that this hand has exactly n of.
    Return None if there is no n-of-a-kind in the hand."""
    from collections import Counter
    c = Counter(ranks)
    for i in set(ranks):
        if c.get(i) == n:
            return i
            break

    return None
```

或

```
def kind(n,ranks):
    "Return the first rank that this hand has exactly n of.Return None
if there is no n-of-a-kind in the hand."
    for r in ranks :
        if ranks.count(r) == n : return r

    return None
```

two_pair

```
def two_pair(ranks):
    """If there are two pair, return the two ranks as a
    tuple: (highest, lowest); otherwise return None."""
    pair=[]
    for i in set(ranks):
        if ranks.count(i) == 2 : pair.append(i)
    if len(pair) == 2:
        return tuple(sorted(pair,reverse = True))
    else:
        return None
```

或

```
def two_pair(ranks):
     "If there are two pair, return the two ranks as a
    tuple: (highest, lowest); otherwise return None."
    pair = kind(2,ranks)
    lowpair = kind(2,list(reversed(ranks)))
    if pair and lowpair ! = pair
        return (pair , lowpair)
    else:
        return None
```

card_ranks

```python
def card_ranks(cards):
    "Return a list of the ranks, sorted with higher first."
    def map(r):
        if r =='T':
            return 10
        elif r == 'J':
            return 11
        elif r == 'Q':
            return 12
        elif r == 'K':
            return 13
        elif r == 'A':
            return 14
        else:
            return int(r)
    ranks = [map(r) for r,s in cards]
    ranks.sort(reverse=True)

    return ranks
```

或

```python
def card_ranks(hand):
    "return a list of the ranks , sorted with higher first"
    ranks = ['--23456789TJQKA'.index(r) for r,s in hand]
    ranks.sort(reverse = True)

return ranks
```

注意到，我们希望A12345 也是straight ，故我们需要对代码进行修改

> 读者可以考虑下，是修改straight 函数呢还是修改card_ranks 函数

我们修改card_ranks 函数得到

```python
def card_ranks(hand):
    "return a list of the ranks , sorted with higher first"
    ranks = ['--23456789TJQKA'.index(r) for r,s in hand]
    ranks.sort(reverse = True)
    return [5,4,3,2,1] if (ranks = [14,5,4,3,2]) else ranks
```

考虑到可能有多个一样的最大，写allmax 函数同时返回多个相同的最大

```
def allmax(iterable ,key = None):
    "return a list of all items equal to the max of the iterable"
    result , maxval =[],None
    key = key or (lambda: x:x)
    for x in iterable:
        xval = key(x)
        if not result or xval > maxval:
            result , maxval = [x] , xval
        elif xval == maxval:
            result.append(x)
    return result
```

于是我们修改poker 函数得到

```
def poker(hands):
    "Return the best hand: poker([hand,...]) => hand"
    return allmax(hands, key=hand_rank)
```

有了规则之后，我们就要进行洗牌和发牌了
首先我们构建一副牌

```
mydeck = [r+s for r in '23456789TJQKA' for s in 'SHDC']
```

然后考虑发牌规则和参赛人数（numhands），poker 默认每人5张牌

```
def deal(numhands, n=5, deck=mydeck):
    random.shuffle(deck)
    hands=[[' ']*n]*numhands
    for i in range(numhands):
        for j in range(n):
            if len(deck)>0:
                hands[i][j]=deck.pop(0)
            else:
                deck=mydeck
                random.shuffle(deck)
    return hands
```

或

```
def deal(numhands , n=5, deck = [r+s for r in '23456789TJQKA' for s in
'SHDC']):
    random.shuffle(deck)
    return [deck[n*i:n*(i+1)] for i in range(numhands)]

# only one deck
```

游戏的组建全部完成，我们可以开始开心地游戏了

```
In [38]: hands = deal(5)

In [40]: hands
Out[40]:
[['TS', '7H', '8D', 'AH', 'TC'],
 ['JS', 'QH', 'QD', '3C', 'KS'],
 ['9S', '8H', 'QS', '2S', '8C'],
 ['2H', '7D', '9D', 'KD', '5H'],
 ['JH', '2D', '3H', '9H', '6D']]

In [39]: poker(hands)
Out[39]: [['JS', 'QH', 'QD', '3C', 'KS']]

# have a look
In [41]: for h in hands:
    ...:     print hand_rank(h)
    ...:
(1, 10, [14, 10, 10, 8, 7])
(1, 12, [13, 12, 12, 11, 3])
(1, 8, [12, 9, 8, 8, 2])
(0, [13, 9, 7, 5, 2])
(0, [11, 9, 6, 3, 2]
```