成都力比科技有限公司-游戏数据分析师笔试题

答题人：杨双杰　　联系方式：17796401944

问题一：

“外星人打算将地球用来种蘑菇，并且已经抓了十个人类……”

外星人用这十个人代表地球60亿人口，将通过外星人的方式来测试这十个人，决定地球是不是有资格加入跨星际委员会，如果没有，就把地球变成一个蘑菇农场。

明天，这十个人将被关在一间漆黑的屋子里前后排成一队，外星人将给每个人戴一顶帽子，帽子为紫色或者绿色，然后外星人会将灯打开，这十个人每个人都无法看见自己头上的帽子是什么颜色，但可以看见排在你前面的每个人头上帽子的颜色。

帽子的颜色是随机的，可能全是紫的，也可能全是绿的，或者是任意的组合。

外星人会从后往前问每一个人:“你头上的帽子是什么颜色？”如果这个人答对了，这个人就安然无事，他所代表的地球上6亿人口也将获救。否则，这个人将被爆头，外星人将把他所代表的6亿人口变成蘑菇的肥料。每个人的答案屋子里所有人都可以听到。

现在，人类的命运在你手上，你可以设计一个方案，使这十个人提前制定一个计划，这个计划必须拯救尽可能多的人。

提示：有个方案可以让你拯救其中至少九个人。

答：方案一

关键点：

通过绿色、紫色帽子的奇偶数来判断。

具体而言：

第一个被询问的人数其余9人帽子的颜色，哪个颜色为奇数，就回答哪个颜色（假设为绿色）。自身存活的概率为50%，此时回答的颜色只是作为奇偶数的一个信号；

第二个被询问的人，继续数其余8人帽子的颜色，如果绿色帽子的颜色数为偶数，则说明自己的帽子为绿色，如果绿色帽子的颜色数为奇数，则说明自己帽子的颜色为紫色；

由于可以听见其他人的回答，后面被询问的人继续按照此方法，数前面人绿颜色帽子数加上已经回答绿色帽子的数量，若为偶数，则自身帽子为绿色，若为奇数，则自身帽子颜色为紫色。

以此类推，直至最后一人。至少可以拯救9人，第一人猜对了，则可以全部存活。

将上述方案，通过`python`代码实现，如下：

```
[2] import numpy as np
```

```
[42] humans_with_hats = np.random.choice(['purple','green'],10)
     humans_with_hats
```

```
array(['green', 'green', 'purple', 'purple', 'purple', 'green', 'green',
       'green', 'purple', 'green'], dtype='<U6')
```

```
[43] def others_reply(first_reply_color,i=1):
         while i<10:
             green_hats_num_after = sum(humans_with_hats[i+1:]==first_reply_color)
             green_hats_num_before = sum(humans_with_hats[1:i]==first_reply_color)
             if first_reply_color=='green':
                 reply_color = 'purple'
                 if (green_hats_num_after+green_hats_num_before) % 2 ==1:
                     print(f'human{i} => hat_color: {humans_with_hats[i]}  reply: {reply_color}')
                 else:
                     print(f'human{i} => hat_color: {humans_with_hats[i]}  reply: {first_reply_color}')

             elif first_reply_color == 'purple':
                 reply_color = 'green'
                 if (green_hats_num_after+green_hats_num_before) % 2 ==1:
                     print(f'human{i} => hat_color: {humans_with_hats[i]}  reply: {reply_color}')
                 else:
                     print(f'human{i} => hat_color: {humans_with_hats[i]}  reply: {first_reply_color}')

             return others_reply(first_reply_color,i+1)

     def humans_reply():
         green_hats_num = sum(humans_with_hats[1:]=='green')
         if green_hats_num % 2 == 1:
             first_reply_color = 'green'
             print(f'human{0} => hat_color: {humans_with_hats[0]}  reply: green')
             others_reply(first_reply_color,i=1)
         else:
             first_reply_color = 'purple'
             print(f"human{0} => hat_color: {humans_with_hats[0]}  reply: purple")
             others_reply(first_reply_color,i=1)


     humans_reply()
```

```
human0 => hat_color: green  reply: green
human1 => hat_color: green  reply: green
human2 => hat_color: purple  reply: purple
human3 => hat_color: purple  reply: purple
human4 => hat_color: purple  reply: purple
human5 => hat_color: green  reply: green
human6 => hat_color: green  reply: green
human7 => hat_color: green  reply: green
human8 => hat_color: purple  reply: purple
human9 => hat_color: green  reply: green
```

答：方案二

关键点：

每个人均回答前一个人帽子的颜色，但要传达出自身帽子颜色与前面一个人帽子颜色是否相同的信息。

具体而言：

第一个人直接回答第二个人帽子的颜色（依然假设为绿色），存活概率为50%；

第二个人知道自己帽子颜色为绿色，同时观察前一个人帽子颜色，若颜色相同，则回答两个字“绿色”，若颜色不同，则只回答一个字“绿”；（或者颜色相同，直接回答“绿色”，颜色不同则回答“不是紫色”）

第三个人根据第二个被询问的人的回答，便知道了自己帽子的颜色，同时通过回答的字数是2还是1向第四个人传达帽子颜色异同的信息；

以此类推，直至最后一个人。至少可以拯救9人，第一人与第二人帽子颜色相同，则可以全部存活。

以`python`代码实现方案二，如下：

```
[45]
def others_reply(pre_reply_color,i=1):

    is_same_color = (len(pre_reply_color.split())==2)

    if i==9:
        if is_same_color:
            last_human_hat_color = pre_reply_color.split()[0]
            print(f'human{i} => hat_color: {humans_with_hats[i]}  reply: {last_human_hat_color}')
        else:
            if pre_reply_color=='green':
                last_human_hat_color = 'purple'
            elif pre_reply_color == 'purple':
                last_human_hat_color = 'green'
            print(f'human{i} => hat_color: {humans_with_hats[i]}  reply: {last_human_hat_color}')


    while i<9:
        if is_same_color:
            current_human_hat_color = pre_reply_color.split()[0]
            if humans_with_hats[i+1]==current_human_hat_color:
                reply_color = current_human_hat_color + ' color'
            else:
                reply_color = current_human_hat_color

            print(f'human{i} => hat_color: {humans_with_hats[i]}  reply: {reply_color}')

        else:
            if pre_reply_color=='green':
                current_human_hat_color = 'purple'
            elif pre_reply_color=='purple':
                current_human_hat_color = 'green'

            if humans_with_hats[i+1]==current_human_hat_color:
                reply_color = current_human_hat_color + ' color'
            else:
                reply_color = current_human_hat_color

            print(f'human{i} => hat_color: {humans_with_hats[i]}  reply: {reply_color}')


        return others_reply(pre_reply_color=reply_color,i=i+1)

def humans_reply():
    if humans_with_hats[1]=='green':
        print(f'human{0} => hat_color: {humans_with_hats[0]}  reply: green color')
        first_reply_color = 'green color'
        others_reply(first_reply_color,i=1)
    else:
        print(f"human{0} => hat_color: {humans_with_hats[0]}  reply: purple color")
        first_reply_color = 'purple color'
        others_reply(first_reply_color,i=1)


humans_reply()
```

```
human0 => hat_color: green  reply: green color
human1 => hat_color: green  reply: green
human2 => hat_color: purple  reply: purple color
human3 => hat_color: purple  reply: purple color
human4 => hat_color: purple  reply: purple
human5 => hat_color: green  reply: green color
human6 => hat_color: green  reply: green color
human7 => hat_color: green  reply: green
human8 => hat_color: purple  reply: purple
human9 => hat_color: green  reply: green
```

问题二：

你成功的从外星人手中解救了所有的人，其中一个人送给你一个机关作为报答，据说解开后可以拿到一张神秘地图。

机关是一个横19竖19的方格棋盘，左上角的格子里有一个棋子，你的任务是将棋子移动到右下角的格子里，棋子每次只能向右或者向下移动一格，不能斜向移动，也不能后退。

只有唯一的一条路径才能打开机关，但是尝试次数是不受限制的。（路径不对的话棋子会自动从右下角回到左上角）

你最多要尝试几次才能找到正确的路径？（从左上角移动到右下角的路径有多少种）

答：方案一

以递归的形式，通过状态转移实现。设方格棋盘为m*n，则从左上角到右下角的路径数`max_paths_num`具有如下状态转移形式：

`max_paths_num(m,n)=max_paths_num(m-1,n)+max_paths_num(m,n-1)`

通过`python`代码实现为：

```
[1] def max_paths_num(m,n):
        if m ==1 or n ==1:
            return 1
        return max_paths_num(m-1,n)+max_paths_num(m,n-1)

[6] %%time
    max_paths_num(18,18)

    Wall time: 11min 29s

    2333606220
```

答：方案二

将移动路径分解，横向移动m步，竖向移动n步，当横向(竖向)移动路径确定后，竖向(横向)路径也就确定了，这可以看作一个排列组合问题，即从m+n-2步中选取m-1或n-1步的组合。即：

$$C_{m+n-2}^{m-1}$$

当m=n=19时，横竖实际移动步数为18步。

对比可见，通过组合的方式计算效率远远大于递归的方式（`56.1ms VS 11min29s`）

```
[-] from scipy.special import comb

[7] %%time
    comb(18+18-2,18-1)

    Wall time: 56.1 ms

    2333606220.0
```

问题三：

解开机关后，发现里面是一所神秘的忍者学院的地址，经过一夜的旅行，你到达了学院门口，却发现前面是一道61环铁索桥。桥上挤满了30个忍者和30个武士，每个人会占满一个环的位置。

忍者在左侧，他们想到学院里去（到右侧）；武士在右侧，他们想出来（到左侧）。但他们都不愿给对方让路。行动规则如下：

（1）当忍者（或武士）前面的铁环为空时，他可以走过去；

（2）当忍者（或武士）前面的铁环有其他人（忍者或武士 ），但是再前面的铁环为空时，他可以跳过去；

（3）忍者已经非常着急了，第一步要安排忍者行动；

（4）忍者不能向左走，武士不能向右走。

现在，你要安排忍者和武士各自的行动顺序，让他们需都能到达对岸。请给出最少的行动步骤数。

根据要求，最终的目的是让忍者与武士"都能到达对岸",即达到双方均能到达对岸的状态即可。

答：最少的行动步数应使得双方最终形成交叉站位

假设忍者与武士的人数均为`n`，`A`表示忍者，`B`表示武士，下划线`_`表示空位。
由于不能后退，过程中中间位置同一方出现相邻站位时则均不能通过，例如[n=3]：

```
`AAB_BBA` 或 `A_BAABB`
```

最少的行动步数应使得双方最终形成交叉站位，即应达到如下状态：

```
`A_BABAB` 或 `ABABA_B`
```

1）当`n=1`时，状态变化及移动步数为：

```
状态          移动步数
`A_B`          -
`A_B`          0
```

2）当`n=2`时，状态变化及移动步数为:

```
状态          移动步数
`AA_BB`        -
`A_ABB`        1
`ABA_B`        2
```

3）当`n=3`时，状态变化及移动步数为:

```
状态          移动步数
`AAA_BBB`      -
`AA_ABBB`      1
`AABA_BB`      2
`AABAB_B`      3
`AAB_BAB`      4
`A_BABAB`      5
```

4）当`n=4`时，状态变化及移动步数为:

状态	移动步数
`AAAA_BBBB`	-
`AAA_ABBBB`	1
`AAABA_BBB`	2
`AAABAB_BB`	3
`AAAB_BABB`	4
`AA_BABABB`	5
`A_ABABABB`	6
`ABA_ABABB`	7
`ABABA_ABB`	8
`ABABABA_B`	9

5) 当`n=5`时，状态变化及移动步数为:

状态	移动步数
`AAAAA_BBBBB`	-
`AAAA_ABBBBB`	1
`AAAABA_BBBB`	2
`AAAABAB_BBB`	3
`AAAAB_BABBB`	4
`AAA_BABABBB`	5
`AA_ABABABBB`	6
`AABA_ABABBB`	7
`AABABA_ABBB`	8
`AABABABA_BB`	9
`AABABABAB_B`	10
`AABABAB_BAB`	11
`AABAB_BABAB`	12
`AAB_BABABAB`	13
`A_BABABABAB`	14

总结上述忍者武士人数`n`与移动步数`steps`的关系为:

```
n=1   =>  steps=0
n=2   =>  steps=2
n=3   =>  steps=5
n=4   =>  steps=9
n=5   =>  steps=14
...   =>  ...
```

设移动步数`steps`服从数列 Y_n , 则有:

$$Y_2 - Y_1 = 2 - 0 = 2 \cdots\cdots\cdots\cdots\cdots\cdots (1)$$

$$Y_3 - Y_2 = 5 - 2 = 3 \cdots\cdots\cdots\cdots\cdots\cdots (2)$$

$$Y_4 - Y_3 = 9 - 5 = 4 \cdots\cdots\cdots\cdots\cdots\cdots (3)$$

$$Y_5 - Y_4 = 14 - 9 = 5 \cdots\cdots\cdots\cdots\cdots\cdots (4)$$

$$\cdots$$

$$Y_{n+1} - Y_n = n + 1 \cdots\cdots\cdots\cdots\cdots\cdots (n)$$

将上述n个式子左边与右边分别相加，有:

$$(Y_{n+1} - Y_n) + (Y_n - Y_{n-1}) + \ldots + (Y_3 - Y_2) + (Y_2 - Y_1)$$

$$= Y_{n+1} - Y_1 = 2 + 3 + 4 + 5 + \ldots + (n+1)$$

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

进而有：

$$Y_{n+1} = \frac{n^2+3n}{2} - Y_1 = \frac{n^2+3n}{2} - 0 = \frac{n^2+3n}{2}$$

$$Y_n = \frac{(n-1)^2+3(n-1)}{2} = \frac{n^2+n-2}{2} \quad (n >= 1)$$

当忍者及武士的人数均为30时，最少的行动步数为：

$$Y_{30} = \frac{30^2+30-2}{2} = 928/2 = 464$$

`python`实现为:(A代表忍者，B代表武士，'_'代表空位)

[1]
```
n = 30
bridge = ["A"]*n + ["_"] + ["B"]*n
print("step 0:",''.join(bridge))
```

```
step 0: AAAAAAAAAAAAAAAAAAAAAAAAAAAA_BBBBBBBBBBBBBBBBBBBBBBBBBBBB
```

[2]
```
steps = 0
m= 2*n+1
While True:
    if  (n%2==1 and bridge[1]=='_') or (n%2==0 and bridge[-2]=='_'):
        break

    for i in range(0,m):
        if  (n%2==1 and bridge[1]=='_') or (n%2==0 and bridge[-2]=='_'):
            break
        try:
            if bridge[i] == "A":
                if bridge[i+1] == "_" and bridge[i+2]=="B":
                   v1 = bridge[i+1]
                   v2 = bridge[i]
                   bridge[i] = v1
                   bridge[i+1] = v2
                   steps += 1
                   print("step %s:" % steps,''.join(bridge))

                if bridge[i+1] =="B" and bridge[i+2] =="_" and bridge[i+3]=="B":
                    v1 = bridge[i+2]
                    v2 = bridge[i]
                    bridge[i] = v1
                    bridge[i+2] = v2
                    steps+=1
                    print("step %s:" % steps,''.join(bridge))

            if bridge[i] == "B":
                if bridge[i-1] == "_" and bridge[i-2]=="A" and bridge[i-3]=="B":
                   v1 = bridge[i-1]
                   v2 = bridge[i]
                   bridge[i] = v1
                   bridge[i-1] = v2
                   steps += 1
                   print("step %s:" % steps,''.join(bridge))

                if bridge[i-1] == "A" and bridge[i-2] =="_" and bridge[i-3]=="A":
                    v1 = bridge[i-2]
                    v2 = bridge[i]
                    bridge[i] = v1
                    bridge[i-2] = v2
                    steps+=1
                    print("step %s:" % steps,''.join(bridge))

        except IndexError:
            pass
steps
```

```
step 445: ABABABABABABABABABA_ABABABABABABABABABABABABABABABABABB
step 446: ABABABABABABABABABBA_ABABABABABABABABABABABABABABABABABB
step 447: ABABABABABABABABABABA_ABABABABABABABABABABABABABABABABB
step 448: ABABABABABABABABABABBA_ABABABABABABABABABABABABABABABABB
step 449: ABABABABABABABABABABABA_ABABABABABABABABABABABABABABABABB
step 450: ABABABABABABABABABABABBA_ABABABABABABABABABABABABABABABABB
step 451: ABABABABABABABABABABABABA_ABABABABABABABABABABABABABABABABB
step 452: ABABABABABABABABABABABABBA_ABABABABABABABABABABABABABABABABB
step 453: ABABABABABABABABABABABABABA_ABABABABABABABABABABABABABABABABB
step 454: ABABABABABABABABABABABABABBA_ABABABABABABABABABABABABABABABABB
step 455: ABABABABABABABABABABABABABABA_ABABABABABABABABABABABABABABABABB
step 456: ABABABABABABABABABABABABABABBA_ABABABABABABABABABABABABABABABABB
step 457: ABABABABABABABABABABABABABABABA_ABABABABABABABABABABABABABABABABB
step 458: ABABABABABABABABABABABABABABABBA_ABABABABABABABABABABABABABABABABB
step 459: ABABABABABABABABABABABABABABABABA_ABABABABABABABABABABABABABABABABB
step 460: ABABABABABABABABABABABABABABABABBA_ABABABABABABABABABABABABABABABABB
step 461: ABABABABABABABABABABABABABABABABABA_ABABABABABABABABABABABABABABABABB
step 462: ABABABABABABABABABABABABABABABABABBA_ABABABABABABABABABABABABABABABABB
step 463: ABABABABABABABABABABABABABABABABABABA_ABABABABABABABABABABABABABABABABB
step 464: ABABABABABABABABABABABABABABABABABABBA_BABABABABABABABABABABABABABABABB
```

问题四：

你进入了学院，并见到了院长，他知道你的事迹之后希望你能先帮他个忙。原来他正打算开一家健身房，想在赚钱的同时捎带推广忍者文化，希望你帮忙制定包年卡的价格。

1. 要满足哪些需求?

答:

- 需求一：开一家健身房，赚钱；
- 需求二：捎带推广忍者文化；
- 需求三：制定年卡价格。

2. 需要收集哪些信息?

答:

- 开健身房 => 市场供求信息；
- 忍者文化 => 客户接受度信息；
- 年卡价格 => 营销策略信息；

具体而言:

1) 健身房的市场供求

主要考虑地点、竞争对手、目标人群健身的意愿。

- 地点因素决定了所面临竞争力的大小、人群集中程度、交通便利性，进而影响获客的难易度；
- 竞争者的数量、提供的服务种类、服务质量等因素将决定自身的存活空间；
- 目标人群对健身的意愿决定了市场对健身的需求。

2) 忍者文化的接受度

目标人群对忍者文化的态度将决定推广活动属于价值创造还是价值减损。

- 如果目标人群对忍者文化非常热爱，通过捆绑与忍者文化相关的服务，可以适当提高年卡的销售价格；
- 如果目标人群对忍者文化无明显偏好，则推广活动只能作为附加服务，不应考虑在年卡价格的制定中；
- 如果目标人群对忍者文化表现出抵触，为了推广忍者文化，则年卡的价格只能采取较低的策略才能吸引用户。

3) 营销策略

营销策略主要解决两个方面的问题：一是面向何种人群提供何种服务实现何种目标；二是成本收益的权衡。

- 面向何种人群：用户群按职业属性可分为学生党、上班组、经商人员、家庭主妇、自由职业等；按年龄段可分为青少年、中年、老年等；按性别分为男、女等；
- 提供何种服务：主提供大众化健身服务，还是针对性别、年龄提供差异化健身服务，抑或是主高端高品质健身服务，都将影响健身卡的价格定位；
- 实现何种目标：如需要快速占领市场，吸纳用户群，则需要以较低的市场价格以吸引客户；如如需要以高品质高口碑逐渐赢得市场，则可以制定相对较高的价格。
- 成本收益估计：在选定了目标群体、提供的服务种类以及要实现的目标后，成本收益的衡量是至关重要的一环。这与健身房的发展周期有关，在前期可能会投入较大的成本，如器材、人员、获客

等；在中期发展稳定时，有了客户基础的情况下，是主要获取收益的阶段；在后期，随着设备的陈旧、更多竞争者的涌入等因素，衰退阶段收益会逐渐降低。

3. 如何推算?

答:

可以简单推算，也可以根据所有收集的信息进行相对复杂的统计测算。

1) 简单测算

根据竞争对手的价格信息、自身的成本投入确定最终的价格。

当然针对本题的需求，还要额外考虑忍者文化的推广带来的影响。

例如，假设有如下信息:

健身房	投入成本(万元)	服务种类	面向人群	销售价格(/人/年)
001	200	大众化服务	所有	2000
002	150	老年健身服务	老年人	1500
003	350	高端健身服务	上班族	4000

忍者学院院长经过一番考虑，准备开一个主要面对忍者学员的健身房，同时又希望能够兼顾学院周边的健身人士，因此，该健身房主要面对学生党，并提供大众化健身服务，那么年卡销售价格应该小于等于2000才具有竞争力:

健身房	投入成本(万元)	服务种类	面向人群	销售价格(/人/年)
忍者学院	200	大众化服务	学生党	<=2000

此外，为了推广忍者文化，根据客户群体对忍者文化的态度，可以分为三个档次:

健身房	投入成本(万元)	服务种类	面向人群	忍者文化接受度	销售价格(/人/年)
忍者学院	200	大众化服务	学生党	热爱	2200
忍者学院	200	大众化服务	学生党	不关心	2000
忍者学院	200	大众化服务	学生党	抵触	1800

2) 统计测算

根据上述市场供求、忍者文化的接受度、营销策略等三个方面可以获得如下信息:

健身房	投入成本	面向人群	服务种类	服务质量	市场方案	发展时期
---	---	---	---	---	---	---

所处地点	人群集中度	交通便利性	竞争对手数量	人群健身的意愿	忍者文化接受度	销售价格

采用一定的统计方法将上述指标量化，然后根据忍者学院院长对健身房的定位，最终可推算出健身房的年卡价格:

- 若数据量很少，可以直接采取分等级加权的方法；
- 若数据量较多，则可以直接采用回归分析方法。

问题五：

1）

已知以下表结构，请用**一条SQL**语句查询出：安装日期在2018-05-01（含）至2018-05-30（含）期间、且最后登录日期在2018-08-05（含）之后的用户总人数，及在满足前述条件的用户中，2018-08-05（含）之后有过付费的总人数、付费的总次数与付费的总金额。

输出结果包括：用户总人数、付费总人数、付费总次数、付费总金额

table_a（每个用户只有一条记录）：uid，install_time（安装日期），last_login_time（最后登录日期）

table_b(日志表，每发生一笔付费产生一条记录)：uid，pay_time，money

例如：查询结果是100,30,60,400

表示，满足安装条件的人数为100，其中有30人在2018-08-05之后有付费，共付费60次，总金额为400$

1）答：

利用python操作整个过程，创建表再模拟了一些数据：

1)

```
[1] import sqlite3
```

```
[2] conn = sqlite3.connect('test.db')
    cursor = conn.cursor()
```

```
[3] # 创建table_a
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS table_a (
            uid BIGINT PRIMARY KEY,
            install_time DATETIME,
            last_login_time DATETIME);
        """)

    # 插入数据
    cursor.execute(
        """
        INSERT INTO table_a
        VALUES
            (10001, '2018-04-15 22:10:32', '2018-06-30 20:56:01'),
            (10002, '2018-05-01 14:32:08', '2018-08-05 16:43:44'),
            (10003, '2018-05-10 10:03:55', '2018-08-15 12:12:23'),
            (10004, '2018-05-20 21:32:43', '2018-08-31 04:56:00'),
            (10005, '2018-05-30 23:11:45', '2018-07-15 23:22:12');
        """
        )

    # 创建table_b
    cursor.execute(
        """
        CREATE TABLE IF NOT EXISTS table_b(
            uid BIGINT,
            pay_time DATETIME,
            money DOUBLE);
        """
        )

    # 插入数据
    cursor.execute(
        """
        INSERT INTO table_b
        VALUES
            (10001, '2018-05-10 22:10:32', 6),
            (10001, '2018-06-06 23:11:45', 10),
            (10003, '2018-08-10 14:32:08', 30),
            (10003, '2018-07-12 23:22:12', 30),
            (10004, '2018-08-10 12:12:23', 30),
            (10004, '2018-08-10 12:12:25', 30);
        """
        )

    conn.commit()
```

下述 SQL 语句便是所要求的“一条SQL”：

```
[4] datas = cursor.execute(
            """
            SELECT
              COUNT(a.uid) AS 用户总人数,
              COUNT(DISTINCT c.uid) AS 付费总人数,
              COUNT(*) AS 付费总次数,
              SUM(money) AS 付费总金额
            FROM
            (
              (SELECT * FROM table_a
              WHERE
                DATE(install_time) BETWEEN '2018-05-01' AND '2018-05-30'
                AND
                DATE(last_login_time) >= '2018-08-05'
              ) a
              INNER JOIN
              (SELECT * FROM table_b
              WHERE
                DATE(pay_time) >= '2018-08-05'
              ) b
              ON a.uid=b.uid
            ) c
            """
        )

datas.fetchall()

[(3, 2, 3, 90.0)]
```

2）PropUse表中含有用户ID和游戏道具使用日期两列数据，Payment表中含有用户ID和付费日期两列数据，示例如下：

PropUse table:

UserID	Dates
10001	2018-06-01
10002	2018-06-01
10003	2018-06-01
…	

Payment table:

UserID	Dates
10002	2018-06-01
10043	2018-06-01
10071	2018-06-02
…	

请写SQL语句，提取出2018年6月份中，所有在某一天只使用游戏道具没有付费的用户ID以及日期

提示：用户在同一天仅发生了使用道具行为没有发生付费行为算作满足条件。

2）答：

```
[5] # 创建 PropUse 表
cursor.execute(
    """
    CREATE TABLE IF NOT EXISTS PropUse (
        UserID BIGINT,
        Dates DATETIME);
    """)

# 插入数据
cursor.execute(
    """
    INSERT INTO PropUse
    VALUES
        (10001, '2018-06-01 22:10:32'),
        (10002, '2018-06-01 14:32:08'),
        (10003, '2018-08-01 10:03:55'),
        (10004, '2018-06-01 21:32:43'),
        (10005, '2018-06-02 23:11:45');
    """
    )

# 创建Payment 表
cursor.execute(
    """
    CREATE TABLE IF NOT EXISTS Payment(
        UserID BIGINT,
        Dates DATETIME);
    """
    )
```

```
# 创建Payment 表
cursor.execute(
    """
    CREATE TABLE IF NOT EXISTS Payment(
        UserID BIGINT,
        Dates DATETIME);
    """
    )

# 插入数据
cursor.execute(
    """
    INSERT INTO Payment
    VALUES
        (10001, '2018-06-01 22:10:32'),
        (10002, '2018-06-01 23:11:45'),
        (10003, '2018-08-10 14:32:08'),
        (10043, '2018-06-01 12:12:23'),
        (10071, '2018-06-02 12:12:25');
    """
    )

conn.commit()
```

下述 SQL 语句便是所要求的：

[8]
```
datas = cursor.execute(
            """
            SELECT UserID,DATE(Dates) FROM PropUse
            WHERE
                DATE(Dates) BETWEEN '2018-06-01' AND '2018-06-30'
            EXCEPT
            SELECT UserID,DATE(Dates) FROM Payment
            WHERE
              DATE(Dates) BETWEEN '2018-06-01' AND '2018-06-30'
            """
        )
datas.fetchall()
```

```
[(10004, '2018-06-01'), (10005, '2018-06-02')]
```

[9]
```
cursor.close()
conn.close()
```