Assignment 3.2 Practice Problem 2 (Split the Bill)

Problem

In the Splitwise app, people form groups and add the expenses of members of the group. This is especially useful for vacations, where people traveling in a group can maintain an account of their expenses and who paid the bills.

All people in the group are assigned distinct IDs between 1 and N, where N is the size of the group.

In addition to keeping a record of the expenditure, Splitwise also calculates the list of shortest-path transfers (defined later) that will settle up all dues.

Each transaction has the following parameters:

- transaction_id - It is a string representing the unique ID by which the transaction is identified.
- paid_by - It is a list of lists, where each element of the list is another list having the form [x, y]. Here, x and y denote that person having ID x paid Rs. y.
- split_as - It is a list of lists, where each element of the list is another list having the form [x, y]. Here, x and y denote that after all dues are settled, a person having ID x will ultimately contribute Rs. y to the transaction.

For any given transaction, the following condition holds true:-

Total_Amt_Paid = Sum_of_all_splits

In other words, the sum total of all amounts in list paid_by equals the sum total of all amounts in list split_as.

Following is the example of a transaction in a group of size N=64:-

- transaction_id : "#f1230"
- paid_by : [[1, 30], [4, 100], [63, 320]]
- split_as : [[1, 120], [2, 20], [3, 40], [4, 40], [37, 100], [51, 40], [53, 90]]

Shortest-Path Transfers: Shortest-path transfers lead to a reduction in the number of transfers.

Specifically, for a group having multiple transactions, the shortest-path transfers will be a list of payments to be made such that:-

- Each payment can be represented by a list of the following form:- [payer_id, payee_id, amount]. There is only 1 payer, and 1 payee in each payment, which are distinct from each other. So, payer_id != payee_id, for any payment.
- Each person (out of the N people) can only either be the payer (in all payments involving him), or the payee, but not both.
- The total amount of money that each person should receive/spend, must be equal to the total amount he would receive/spend according to the given list of transactions. Clearly, there can be several shortest-path transfers for a particular list of transactions.

Specifically, the lexicographically smallest shortest path has the following:-

- Arrange people who have borrowed money in ascending order of their IDs. Do the same for people who have lent money.
- Now, construct payments so that the least borrower ID has to pay the least lender ID. Continue this process, till all debts have been settled.

Task

Given N members in a group, and lists representing the transactions(expenses), print the payments involved in the lexicographically smallest shortest-path transfers for the group.

Example

Input:

- N = 4
- 5 transactions, that can be represented as follows:-
  transaction_id = "#a1234", paid_by = [[1, 60]], split_as = [[2, 60]].
  transaction_id = "#a2142", paid_by = [[2, 40]], split_as = [[3, 40]].
  transaction_id = "#b3310", paid_by = [[3, 30]], split_as = [[4, 30]].
  transaction_id = "#b2211", paid_by = [[4, 30]], split_as = [[3, 30]].
  transaction_id = "#f1210", paid_by = [[3, 20]], split_as = [[1, 20]].

Output:

- 2 payments (of the form [payer_id, payee_id, amount]) are to be made, represented by the list:-
  - [[1, 2, 20], [1, 3, 20]]

Approach:

- The given list of payments satisfies all three necessary conditions. Hence, it is a Shortest-Path Transfer.

Function description

Complete the function solve. This function takes the following 2 parameters and returns the required answer:

- N: An integer, representing the number of people in the group.
- transaction_list: A list (vector) of transactions. Each transaction is a dictionary, having keys “transaction_id”, “paid_by” and “split_as”. (The contents of each transaction are explained above)

Input format

Note: This is the input format that you must use to provide custom input (available above the Compile and Test button).

- The first line contains two space-separated integers N and M, the number of people in the group, and the number of transactions recorded.
- The next lines describe the M transactions as follows:-
- Each new transaction begins from a new line.
- The first line of each transaction contains a string, representing the transaction_id of the transaction.
- The 2nd line of each transaction contains 2 space-separated integers n_payers and n_splits.
- n_payers denotes the number of people in the paid_by list. n_splits denotes the number of people in the split_as list.
- The next n_payers lines contain two space-separated integers, the payer and the amount paid.
- The next n_splits lines contain two space-separated integers, the borrower and the amount borrowed.

Output format

Print the answer in the given format.

- In the first line, print a single integer K, denoting the number of payments involved in the Shortest Path Transfer.

- The next K lines should represent the K payments. Each payment should be printed in a single line as 3 space-separated integers payer_id, payee_id, and amount. Here, payer_id is the ID of the person who needs to pay the amount of money to the person with ID payee_id.

Constraints

- $2 \leq N \leq 2 * 10^5$
- $1 \leq M \leq 5000$
- $1 \leq len(transaction[i][paid\_by]) + len(transaction[i][split\_as]) \leq 50$
- $1 \leq total\_money\_exchanged\_in\_each\_transaction \leq 10^7$

**SAMPLE **

Input	Output

6 5
#itsmylife
2 3
1 25
3 15
4 10
5 25
6 5
#itsnow
1 4
4 100
1 25
2 25
3 25
4 25
#ornever
2 2
5 30
3 10
1 25
4 15
#iaintgonna
1 3
2 150
1 50
2 50
3 50
#liveforever
2 2
5 13
6 25
4 25
1 13
	1 2 75
	1 4 13
	3 4 12
	3 5 18
	3 6 20

```
# CREATED & ANSWERED By kurtymittens (Kurt Russel Villamor)
```

ANSWER

- **for this problem, i suppose that i can solve it by creating a dictionary. But since we aim for an optimized solution, I think the use of arrays are cool.**
- **Using the sample input/output as a guide. I think the implementation of graphs are not really needed but since this problem is a set of ordered pairs, this can be theoretically a representation of graphs**

ALGORITHM

START

1. **Get user Input for Transaction ID**
2. **Get the User input for payer and payee**
3. **split the the string inputs**
4. **transform to be int**
5. **Create balance array**
6. **edit the Balance array per iterations**
7. **Make the values inside the array 0** --> (HOW??)
8. Present Output
   END

- **With this guided now so algo we can create an pseudocode**

PSEUDOCODE

```
user input --> numberOfPeople, numberOfTransactions
balance[base of noOfPeople]
- loop through numberOfTransactions
    - input transactionID
    - input noOfPayer, noOfPayee
    - input peopleId, price
        - if payer, add to balance
        - if payee, negate to balance
- loop through balance[]
    - if positive number
    - make it zero by adding negative numbers in the balance[]
- print result
```

CODING

- with that we can do some codes. this line allows the user for double integer inputs that seperated with whitespace

```
group , transactions = input().split()
```

```
6 5
```

- For the recording/noting purposes I include and dictionary that shows the transaction data neatly

```
dictTransactions = {} # for recording/representation purposes
```

- Based on the number of members of the group, I made a Balance array. this will be usefull for tracking the money lent(positive) and the money need to be paid(negative)

```
balance = [0 for i in range(int(group))] # for storing balances in just one array
```

- This part of the asks for the inputs of the following: Transaction ID(string), number of payers and payees (both integers), id's of the payer or payees and their corresponding price of their contribution(which also the integers processed to balance array).

```
for i in range(int(transactions)):
  transacID = input() # The transaction ID
  dictTransactions[transacID] = [["paid_by"], ["split_by"]] # added for representation purposes
  payers, payees = input().split()

  for k in range(int(payers)):
    payerId, price1 = input().split() # The transaction ID
    dictTransactions[transacID][0].append([payerId, price1]) # added for representation purposes
    balance[int(payerId)-1] += int(price1)

  for l in range(int(payees)):
    payeeId, price2 = input().split()
    dictTransactions[transacID][1].append([payeeId, price2]) # added for representation purposes
    balance[int(payeeId)-1] -= int(price2)
```

```
#NeverGonnaGiveyouUp
2 3
1 25
3 15
4 10
5 25
6 5
#NeverGonnaLetYouDown
1 4
4 100
1 25
2 25
3 25
4 25
#NeverGonnaRunAround
2 2
5 30
3 10
1 25
```

```
4 15
#AndDessertYou
1 3
2 150
1 50
 2 50
3 50
# NeverGonnaMakeYouCry
2 2
5 13
6 25
4 25
1 13
```

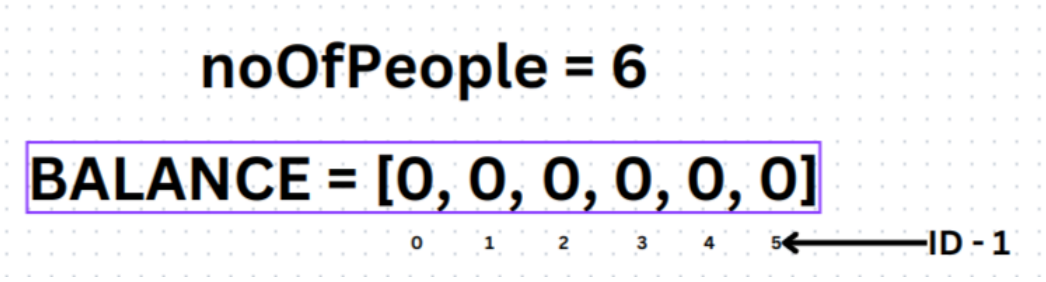
- Lets check the transactions first:

```
print(dictTransactions)
```

```
{'#NeverGonnaGiveyouUp': [['paid_by', ['1', '25'], ['3', '15']], ['split_by', ['4', '10'], ['5', '25'], ['6', '5']]], '#NeverGonnaLetYouDown': [['paid_by', ['4', '100']], ['split_by', ['1', '25'], ['2', '25'], ['3
```

LOGIC FOR THE BALANCE ARRAY

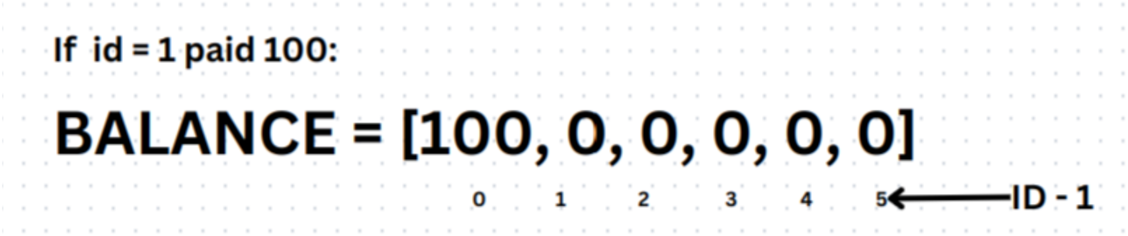
- Recall of the balance array, this array uses its index to record the price the members lent or paid



noOfPeople = 6

BALANCE = [0, 0, 0, 0, 0, 0]

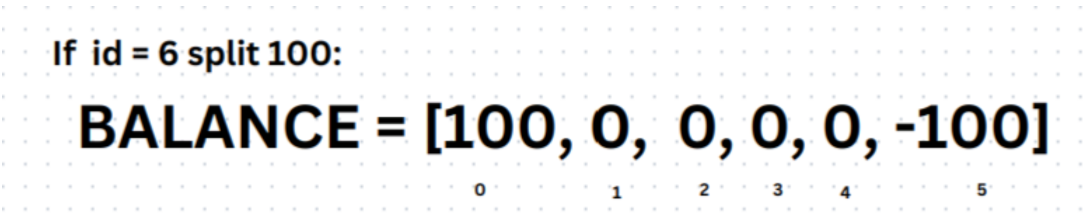
- for example, if id=1 is the one who pays, index 0 (since id - 1) will have an additional price



If id = 1 paid 100:

BALANCE = [100, 0, 0, 0, 0, 0]

- but if the id/person is one of the one who split, it will be negated in the array



If id = 6 split 100:

BALANCE = [100, 0, 0, 0, 0, -100]

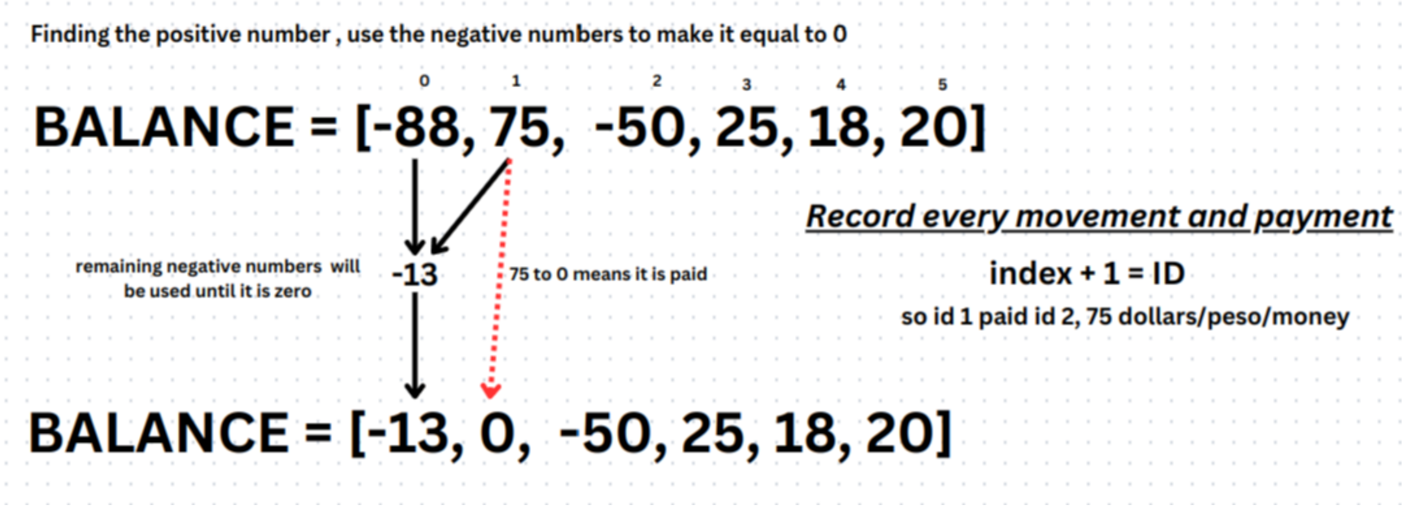
- With these we can easily record the prices in Faster way.For reference,we can see here the balance array after n iterations.

```
print(balance)
```

```
[-88, 75, -50, 25, 18, 20]
```

Lexicographically Smallest Shortest Path

- ths section process the balance array and record the processes for the output showing the shortest way to pay all the balance.



Finding the positive number , use the negative numbers to make it equal to 0

BALANCE = [-88, 75, -50, 25, 18, 20]

BALANCE = [-13, 0, -50, 25, 18, 20]

Record every movement and payment

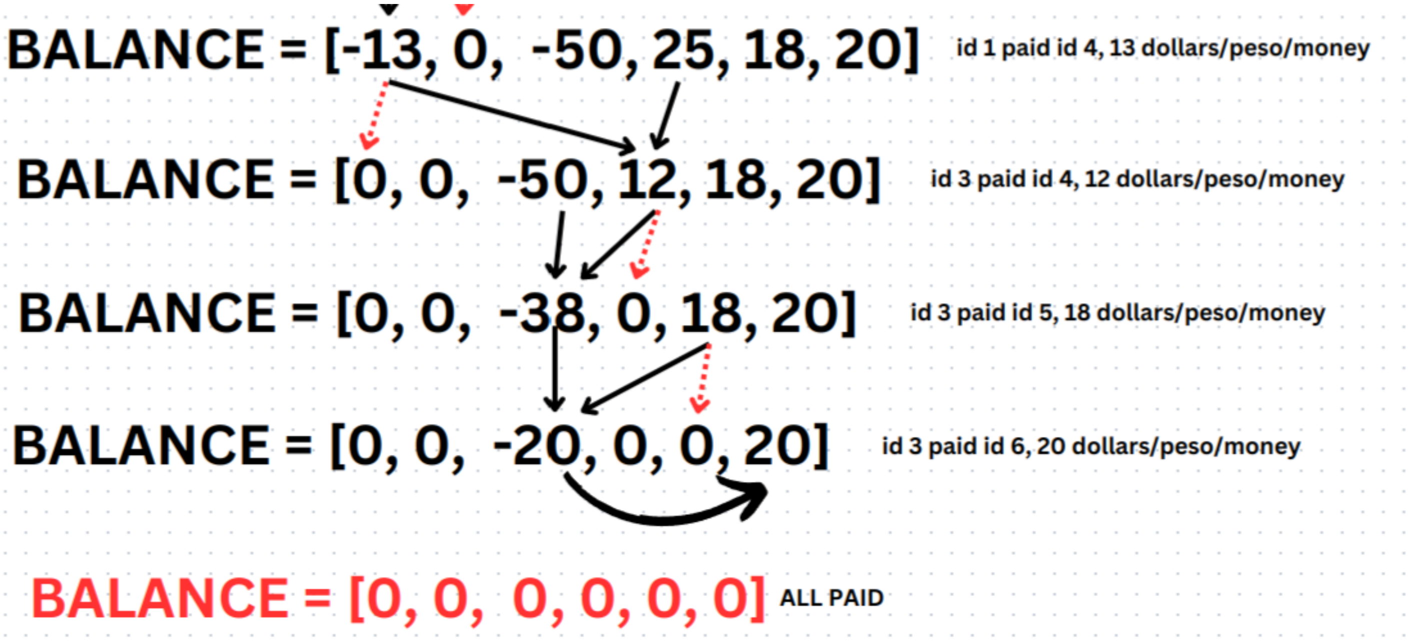
index + 1 = ID

so id 1 paid id 2, 75 dollars/peso/money

- this representation how the balance array is processed.

ALGORITHM:

1. FIND POSITIVE NUM
2. FIND NEGATIVE NUM
3. ADD NEGATIVE TO POSITIVE NUM
4. RECORD
5. REPEAT 3
6. IF POSITIVE NUM = 0:
7. BACK TO 1



BALANCE = [-13, 0, -50, 25, 18, 20] id 1 paid id 4, 13 dollars/peso/money

BALANCE = [0, 0, -50, 12, 18, 20] id 3 paid id 4, 12 dollars/peso/money

BALANCE = [0, 0, -38, 0, 18, 20] id 3 paid id 5, 18 dollars/peso/money

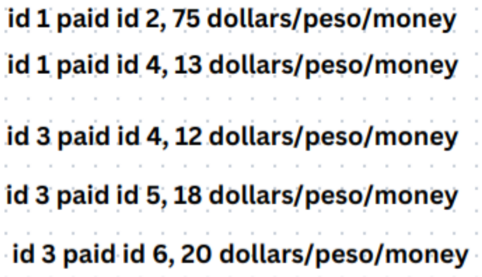
BALANCE = [0, 0, -20, 0, 0, 20] id 3 paid id 6, 20 dollars/peso/money

BALANCE = [0, 0, 0, 0, 0, 0] ALL PAID

- If balance array elements are all 0 again, then all are paid.

```
# CODE
answer = []
for bal in range(len(balance)):
  index = 0
  if balance[bal] > 0: # gets only positives
    while balance[bal] ≠ 0:
      if index == bal: # disregards same index
        index += 1
        continue
      if balance[index] < 0:
        min_balance = min(balance[bal], abs(balance[index]))
        balance[bal] -= min_balance
        balance[index] += min_balance
        answer.append([index+1, bal+1, min_balance])
      index += 1
```

OUTPUT



id 1 paid id 2, 75 dollars/peso/money

id 1 paid id 4, 13 dollars/peso/money

id 3 paid id 4, 12 dollars/peso/money

id 3 paid id 5, 18 dollars/peso/money

id 3 paid id 6, 20 dollars/peso/money

```
print(answer)
```

```
[[1, 2, 75], [1, 4, 13], [3, 4, 12], [3, 5, 18], [3, 6, 20]]
```

```
for i in answer: # SAME FOR THE REQUIRED OUTPUT
  for j in i:
    print(j, end = " ")
  print()
```

```
1 2 75
1 4 13
3 4 12
3 5 18
3 6 20
```

END