Share It

There are some friends living in a hostel who share many things among them daily. They are very economical and never want to waste a penny. Whenever they need something which they can share among them to save money, they do. But the real problem is: whenever they buy some item which they'll share among them, it's very difficult to share the price of the item among them instantly e.g. some item costs 10 bucks and to be shared among 3 friends, each friend gets a share of 3.33 bucks to pay, it's difficult to share it instantly, isn't it? I mean all of them may not have exactly 3.33 bucks in their wallet to pay for settlement. Assume they always pay in cash and share the price of the item equally among them.

So, all the friends come up with below idea to ease their lives:

"Every time we buy some item which is going to be shared among some or all of us, we need not share its price while buying it. Instead one or some of us will pay it while buying and record it for settlement at the end of month."

Basically, they need a software application where *"they can record the purchase of a shared item with its name, purchase date, price, who paid how much while buying it and among whom the item is going to be shared. At the end of the month, the application should be able to find out, who owes how much to whom."*

So far, so good. But...

1. At the end of the month, the application should find out the minimum number of transactions needed among friends for settlement. Say, A needs to pay 10 bucks to B (A->B: 10) and B needs to pay 5 bucks to A (B->A: 5), then A can just pay 5 bucks to B (A->B: 5) and so only 1 transaction is enough for settlement. Similarly, if A->B: 10 and B->C: 10, then A->C: 10 (only 1 transaction is enough again).
2. Sum of the amounts in all transactions in final settlement should be minimum. Say, A->B: 10 and B->C: 5 (sum of the amounts = 10+5 = 15), then A->B: 5 and A->C: 5 (sum of the amounts = 5+5 = 10) should be the transactions in final settlement.

Input

- First line of Input will contain the number of test cases **T (1 <= T <= 100)**. **T** test cases follow.
- First line of each test case will contain two integers **N** and **S**, the number of friends **N (1 <= N <= 100)** numbered from 1 to N and the number of transactions **S (1 <= S <= 1000)** amongst friends within a month.
- **2^{nd} to $(S+1)^{th}$** lines of each test case will contain the descriptions of all **S** transactions, one transaction per line in this format (all values are space separated, angular brackets are for clarification only): **<F> <A> <B_1> <B_2> ... <B_N>** where
  - **F (1<= F <= N):** serial number of a friend,
  - **A (0.01 <= A <= 10000.00):** amount with 2 decimal places paid by **F** in this transaction and
  - **B_i (0 or 1):** 0 if the amount **A** is not to be shared with friend numbered **i**, 1 if it is to be shared with friend numbered **i**. Amount is shared with at least one of the friends. *(Note: if amount **A** is to be shared among **n** friends, truncate each share i.e. **A/n** to 2 decimal places. e.g. if A=10.00 and n=3, then A/n=3.33, similarly if A=20.00 and n=3, then A/n=6.66)*

Output

For each test case, find a solution with minimum number of transactions in final settlement, with sum of the amounts involved in those transactions being minimum (call it **MINIMUM_SUM_AMOUNTS**).

- Output a single line for each test case containing the **MINIMUM_SUM_AMOUNTS** with 2 decimal places only.

Example

Input:

```
2
2 2
1 5.00 0 1
2 10.00 1 0
3 2
2 10.00 1 0 0
3 5.00 0 1 0
```

Output:

```
5.00
10.00
```

Explanation:

1. For 1st testcase, friend #1 will pay 5.00 bucks to friend #2 in the final settlement. (MINIMUM_SUM_AMOUNTS = 5.00)
2. For 2nd testcase, friend #1 will pay 5.00 bucks each to friend #2 and friend #3 in the final settlement. (MINIMUM_SUM_AMOUNTS = 5.00 + 5.00 = 10.00)