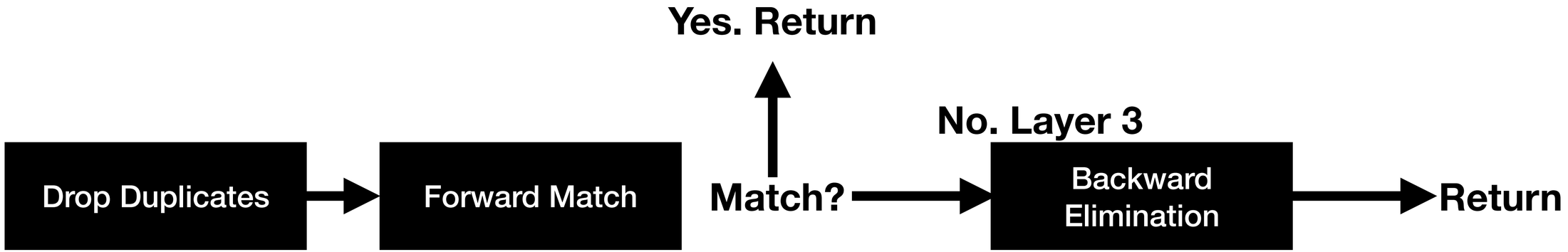
Algorithm

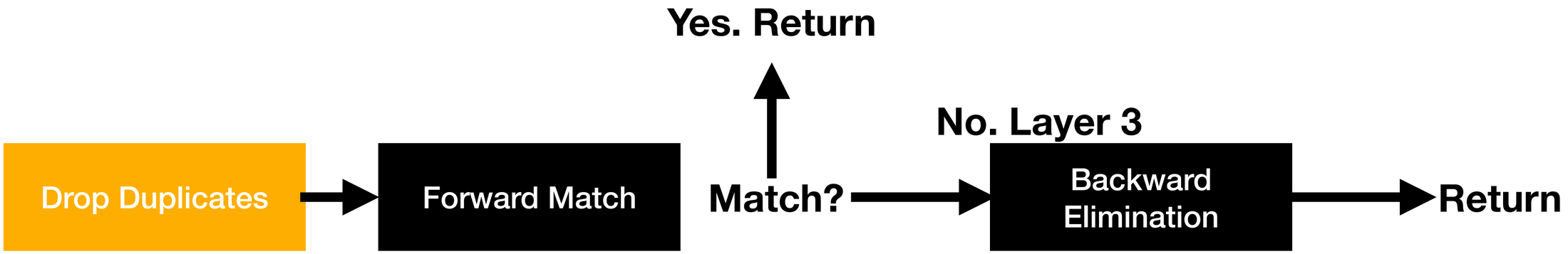
Reconciliation Account

2021/01/27

Three Layers

- Drop Duplicates
- Forward Match
- Backward Elimination




Yes. Return
No. Layer 3
Drop Duplicates
Forward Match
Match?
Backward Elimination
Return

Layer 1: Drop Duplicates

- **Single Drop**
- Merge Drop
- Reversed Merge

The simple duplicate values will be dropped in this step.

Internal Credit
100
~~40~~
70
…

Match!

External Debit
99
1
~~40~~
…

Layer 1: Drop Duplicates; Pseudo Code

- **Single Drop**
- Merge Drop
- Reversed Merge

```
while True:
  update = False
  for amount_x in internal_account:
    for amount_y in external_account:
      If amount_x == amount_y:
        internal_account = internal_account.exclude(amount_x)
        external_account = external_account.exclude(amount_y)
        update = True
        break
    If update:
      break
  If not update:
    break
```

Layer 1: Drop Duplicates

- Single Drop
- **Merge Drop**
- Reversed Merge Drop

This step will create a temp column to store the accumulate sum in the first column.

Internal Credit	Accumulate Sum (temp)
100	100
~~40~~	X
70	170
…	…

External Debit	Accumulate Sum (temp)
99	99
1	100
~~40~~	X
…	…

Layer 1: Drop Duplicates

- Single Drop
- **Merge Drop**
- Reversed Merge Drop

This step will create a temp column to store the accumulate sum in the first column.

Internal Credit	Accumulate Sum (temp)
100	100
~~40~~	X
70	170
…	…

External Debit	Accumulate Sum (temp)
99	99
1	100
~~40~~	X
…	…

Layer 1: Drop Duplicates

- Single Drop
- **Merge Drop**
- Reversed Merge Drop

Then start matching from the top. If the same value is reached, drop all the values above.

Internal Credit	Accumulate Sum (temp)
100	100
~~40~~	X
70	170
…	…

Match!

External Debit	Accumulate Sum (temp)
99	99
1	100
~~40~~	X
…	…

Layer 1: Drop Duplicates

- Single Drop
- **Merge Drop**
- Reversed Merge Drop

Then start matching from the top. If the same value is reached, drop all the values above.

Internal Credit	Accumulate Sum (temp)
~~100~~	100
~~40~~	X
70	170
…	…

External Debit	Accumulate Sum (temp)
~~99~~	99
~~1~~	100
~~40~~	X
…	…

Layer 1: Drop Duplicates; Pseudo Code

- Single Drop
- **Merge Drop**
- Reversed Merge Drop

```
internal_account.add_col(cumsum)
external_account.add_col(cumsum)
while True:
    update = False
    for i_idx in internal_account:
        for e_idx in external_account:
            If internal_account[i_idx, cumsum] == external_account[e_idx, cumsum]:
                internal_account = internal_account[i_idx + 1:, :]
                external_account = internal_account[e_idx + 1:, :]
                update = True
                break
        If update:
            break
    If not update:
        break
```

Layer 1: Drop Duplicates

- Single Drop
- Merge Drop
- **Reversed Merge Drop**

This step will reverse the data frame and conduct merge drop again.

Internal Credit	Accumulate Sum (temp)
39	39
1	40
15	55
…	…

External Debit	Accumulate Sum (temp)
20	20
20	40
40	80
…	…

Layer 1: Drop Duplicates

- Single Drop
- Merge Drop
- **Reversed Merge Drop**

This step will reverse the data frame and conduct merge drop again.

Internal Credit	Accumulate Sum (temp)
39	39
1	40
15	55
…	…

External Debit	Accumulate Sum (temp)
20	20
20	40
40	80
…	…

Layer 1: Drop Duplicates

- Single Drop
- Merge Drop
- **Reversed Merge Drop**

This step will reverse the data frame and conduct merge drop again.

Internal Credit	Accumulate Sum (temp)
39	39
1	40
15	55
…	…

External Debit	Accumulate Sum (temp)
20	20
20	40
40	80
…	…

Layer 1: Drop Duplicates

- Single Drop
- Merge Drop
- **Reversed Merge Drop**

This step will reverse the data frame and conduct merge drop again.

Internal Credit	Accumulate Sum (temp)
39	39
1	40
15	55
…	…

External Debit	Accumulate Sum (temp)
20	20
20	40
40	80
…	…

Layer 1: Drop Duplicates

- Single Drop
- Merge Drop
- **Reversed Merge Drop**

This step will reverse the data frame and conduct merge drop again.

Internal Credit	Accumulate Sum (temp)
39	39
1	40
15	55
…	…

Match!

External Debit	Accumulate Sum (temp)
20	20
20	40
40	80
…	…

Layer 1: Drop Duplicates

- Single Drop
- Merge Drop
- **Reversed Merge Drop**

This step will reverse the data frame and conduct merge drop again.

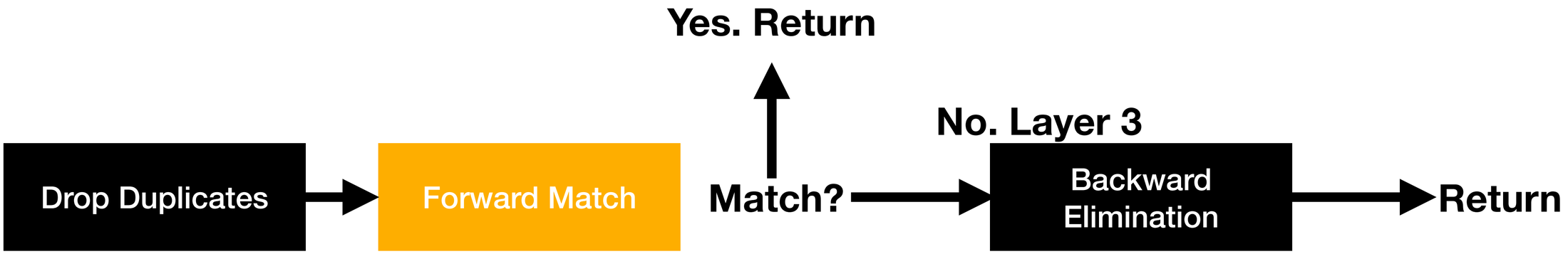
Internal Credit	Accumulate Sum (temp)
~~39~~	39
~~1~~	40
15	55
…	…

External Debit	Accumulate Sum (temp)
~~20~~	20
~~20~~	40
40	80
…	…

Layer 1: Drop Duplicates; Pseudo Code

- Single Drop
- Merge Drop
- **Reversed Merge Drop**

```
internal_account.reverse()
external_account.reverse()
internal_account.add_col(cumsum)
external_account.add_col(cumsum)
while True:
    update = False
    for i_idx in internal_account:
        for e_idx in external_account:
            If internal_account[i_idx, cumsum] == external_account[e_idx, cumsum]:
                internal_account = internal_account[i_idx + 1:, :]
                external_account = internal_account[e_idx + 1:, :]
                update = True
                break
        If update:
            break
    If not update:
        break
```


Yes. Return
No. Layer 3
Drop Duplicates
Forward Match
Match?
Backward Elimination
Return

Layer 2: Forward Match

A helper function will be used to determine the optimal selection of the combination of two array.

Currently, the number is set to 1000 per array so that the maximum operation will not exceed 1000 * 1000 + 1000.

Internal Credit
1000
20
70
…

External Debit
20
1
40
…

Layer 2: Forward Match; Pseudo Code

A helper function will be used to determine the optimal selection of the combination of two array.

Currently, the number is set to 1000 per array so that the maximum operation will not exceed 1000 * 1000 + 1000.

The mathematical formula for optimization is:

$$\max(x) \; subject \; to \sum_{i=1}^{x} \binom{n}{i} < 1000$$

Where x is the optimal number of choose; n is the total length of the input array.

```
def helper_func(array):
    num = len(array)
    counter = 0
    limit = 1000

    for n in num:
      counter += combination(num, n)
      If counter > limit:
        return max(n - 1, 1)
```

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Internal Credit
1000
20
70
…

External Debit
20
1
40
…

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square

Internal Credit
1000
20
70
…

External Debit
20
1
40
…

Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square

Internal Credit
1000
20
70
…

External Debit
20
1
40
…

Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square

Internal Credit
1000
20
70
…

External Debit
20
1
40
…

Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square

Internal Credit
1000
20
70
…

External Debit
20
1
40
…

Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square

Internal Credit
1000
20
70
…

External Debit
20
1
40
…

Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square

Internal Credit
1000
20
70
…

External Debit
20
1
40
…

Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square

Internal Credit
1000
20
70
…

External Debit
20
1
40
…

Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square

Internal Credit	External Debit
1000	20
20	1
70	40
…	…

Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square

Internal Credit
1000
20
70
…

External Debit
20
1
40
…

Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

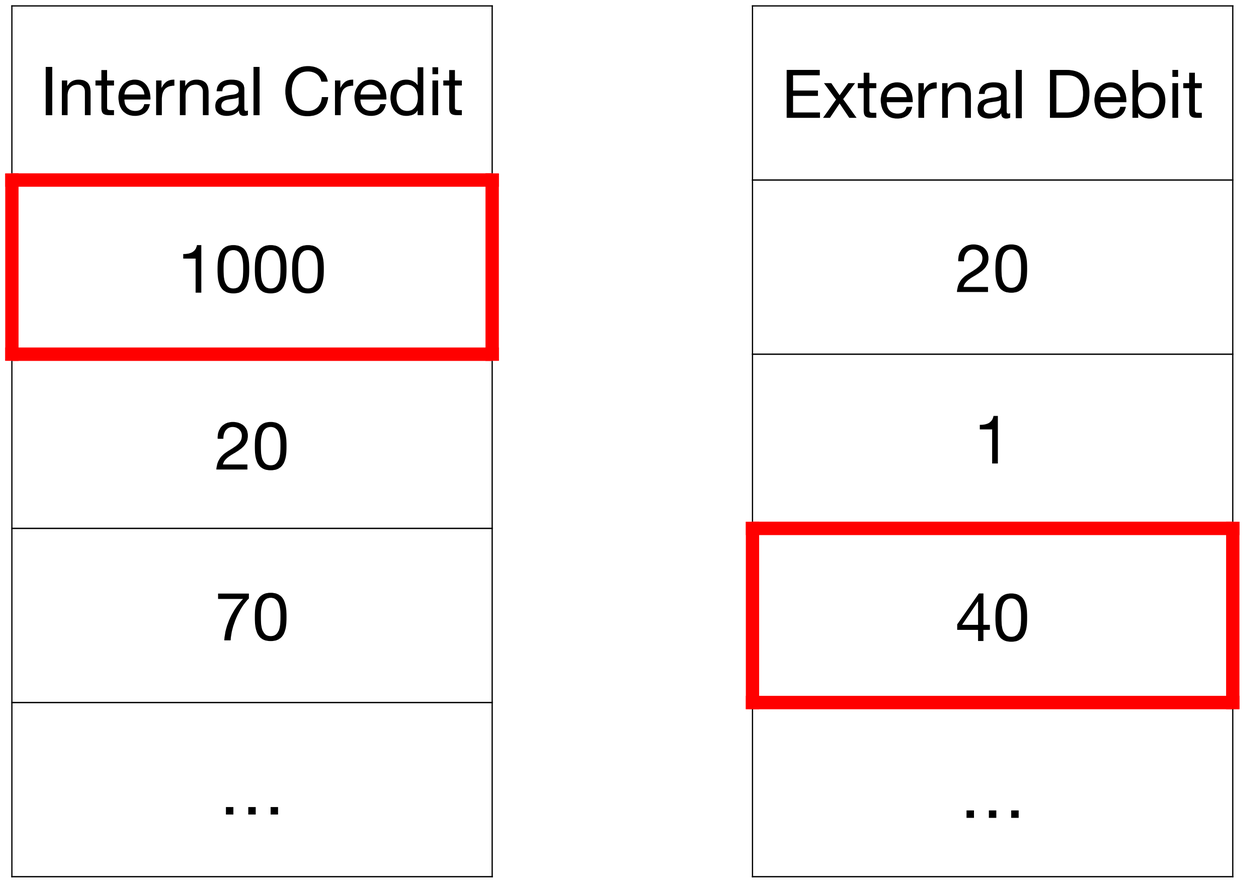
In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square



Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square

Internal Credit
1000
20
70
…

External Debit
20
1
40
…

Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

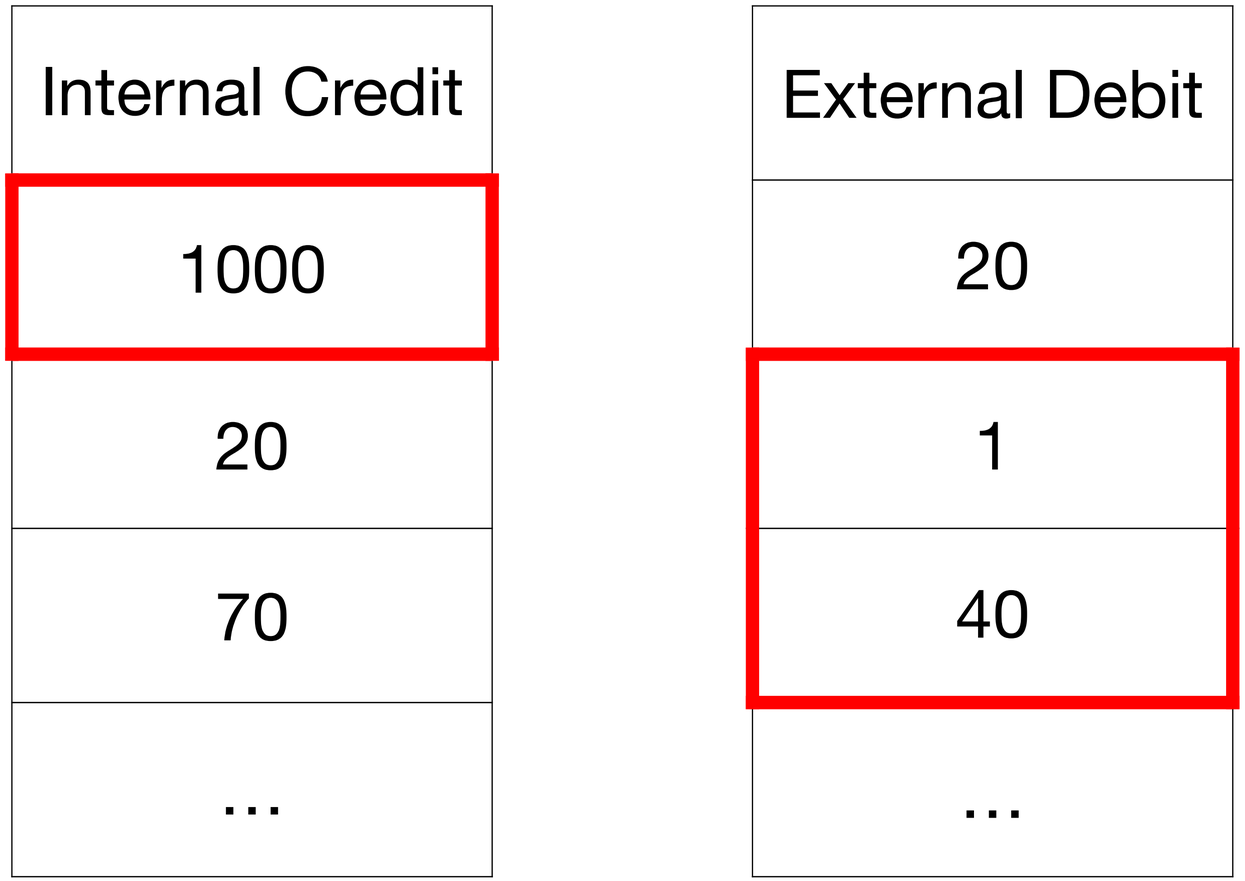
In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square



Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

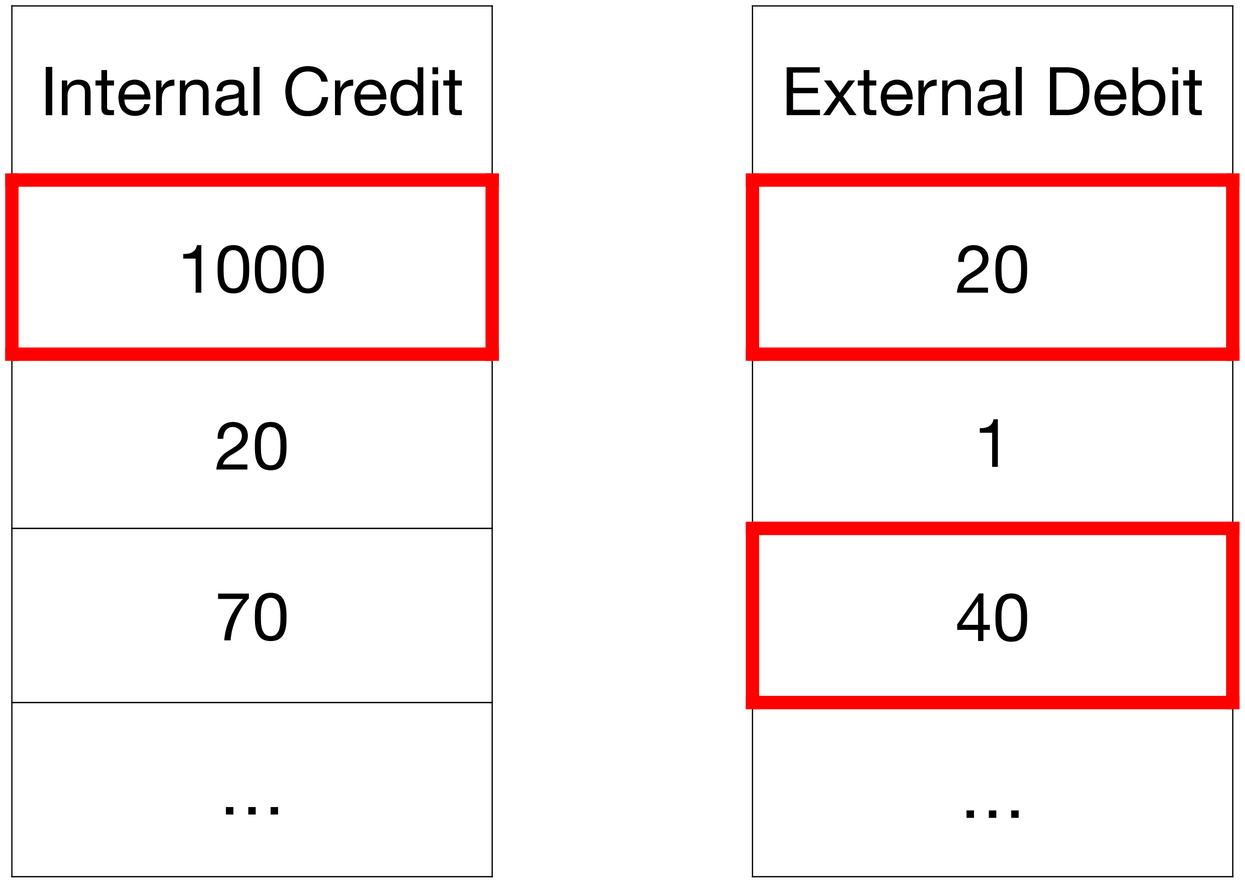
In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square



Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

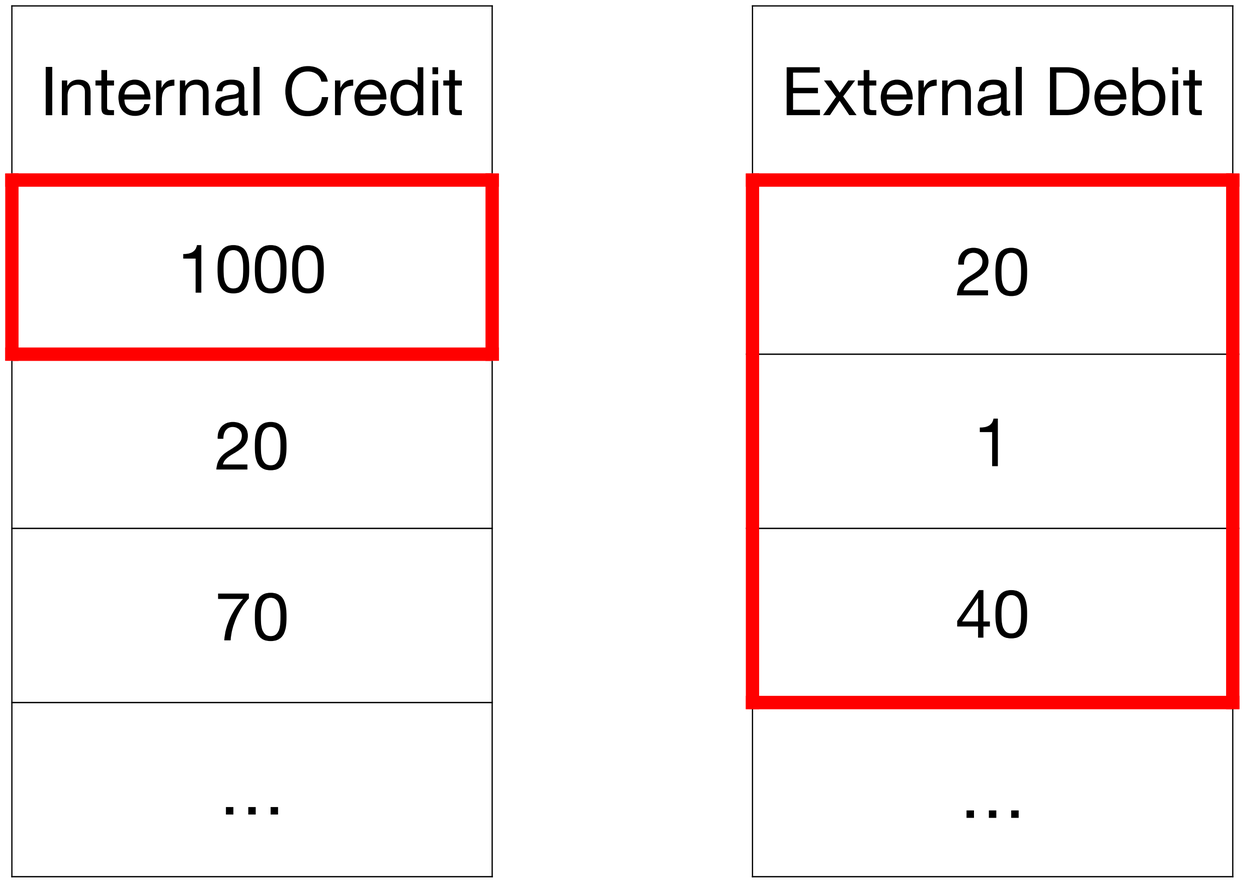
In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square



Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

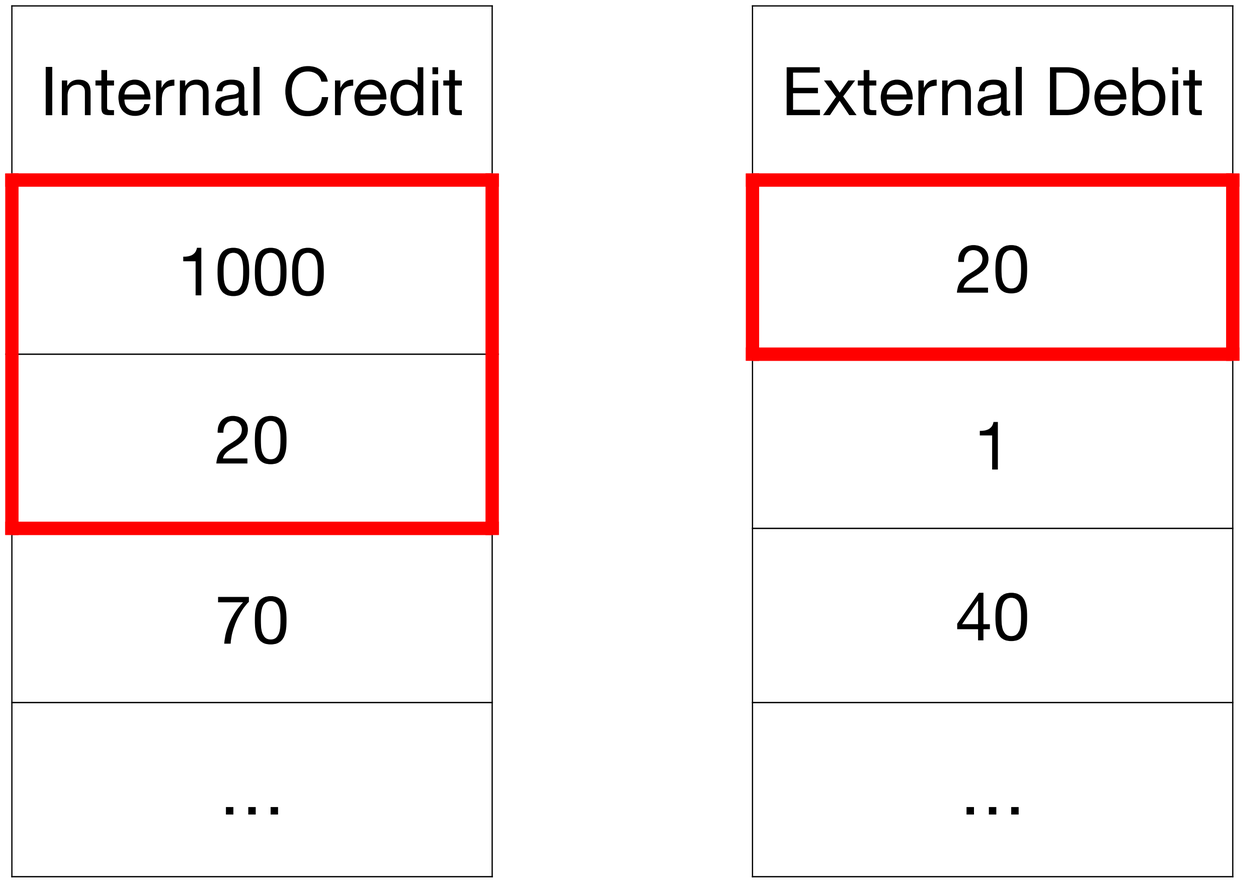
In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square



Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

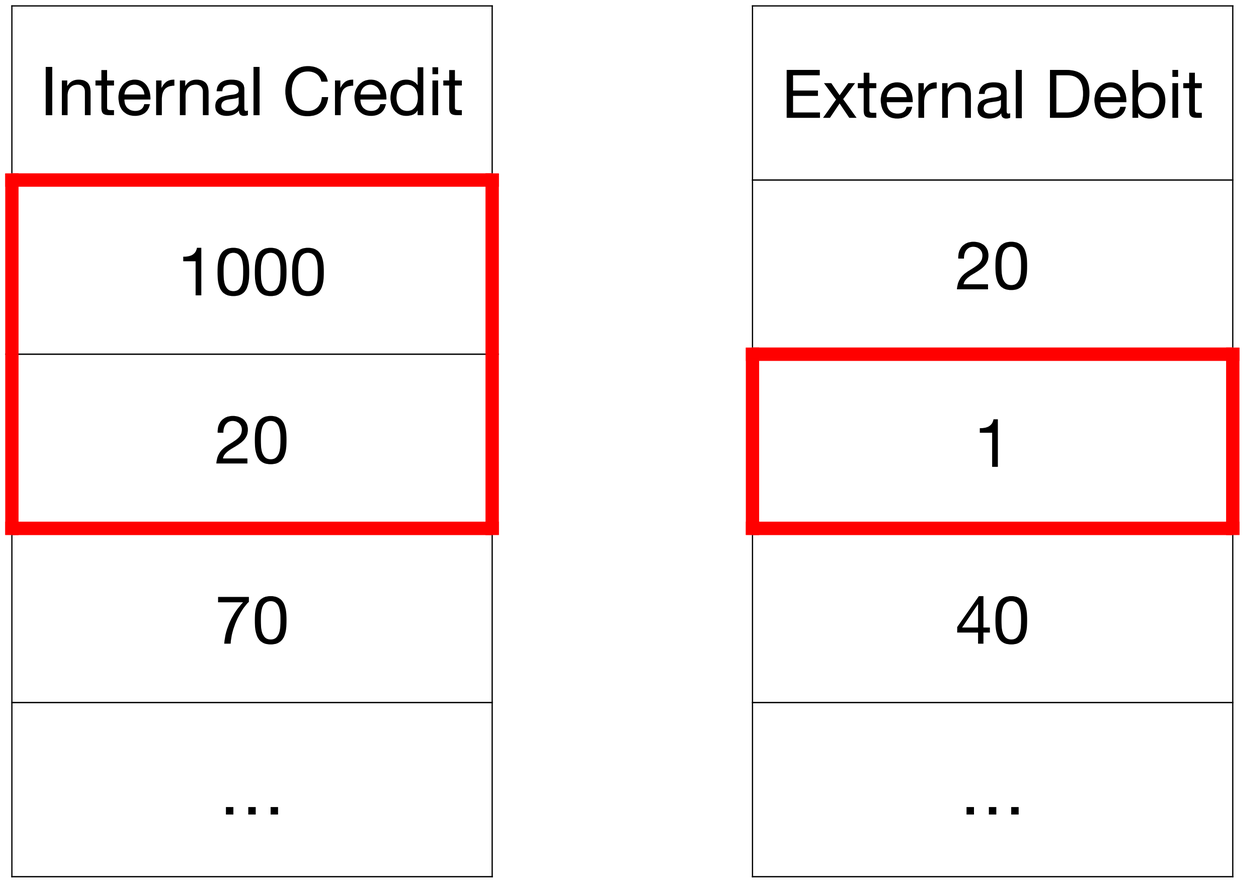
In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square



Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

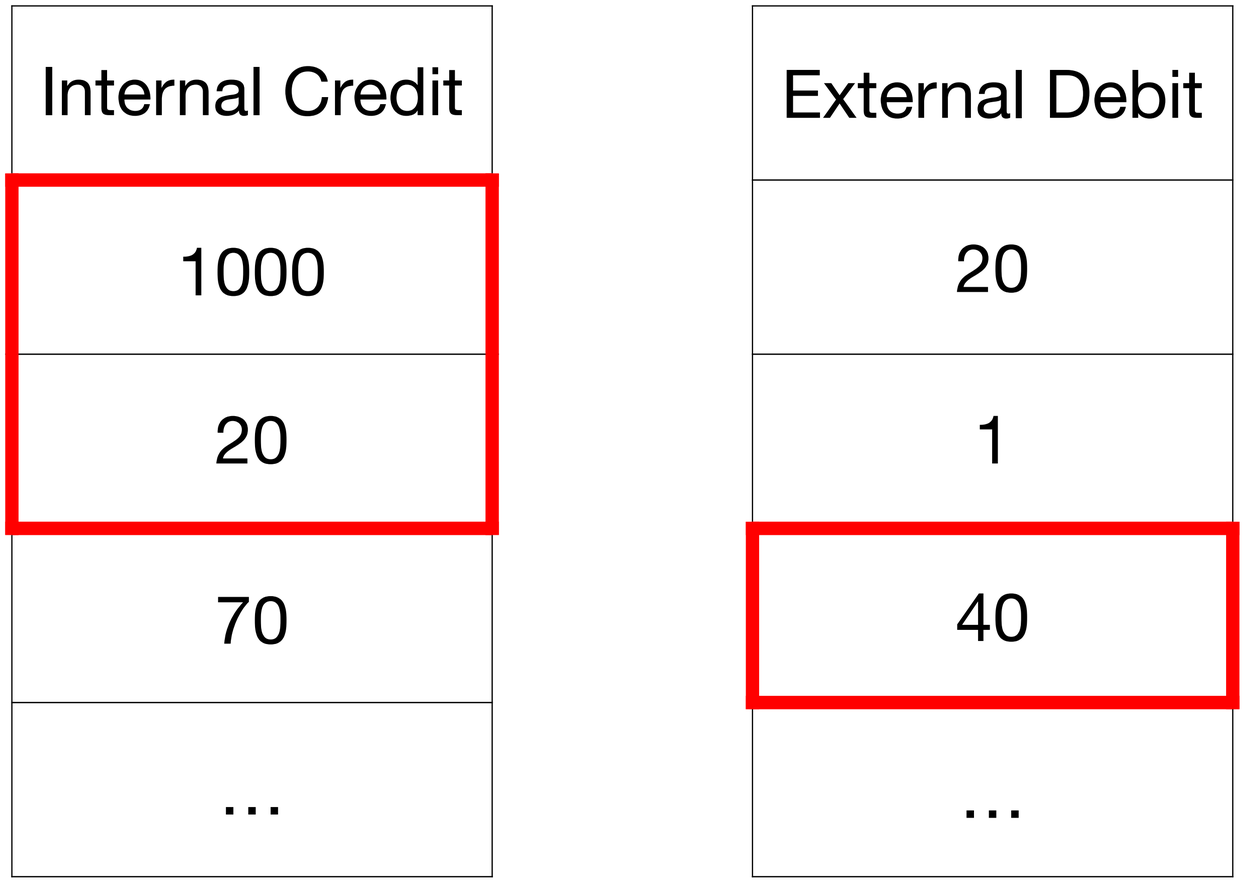
In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square



Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square

Internal Credit
1000
20
70
…

External Debit
20
1
40
…

Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

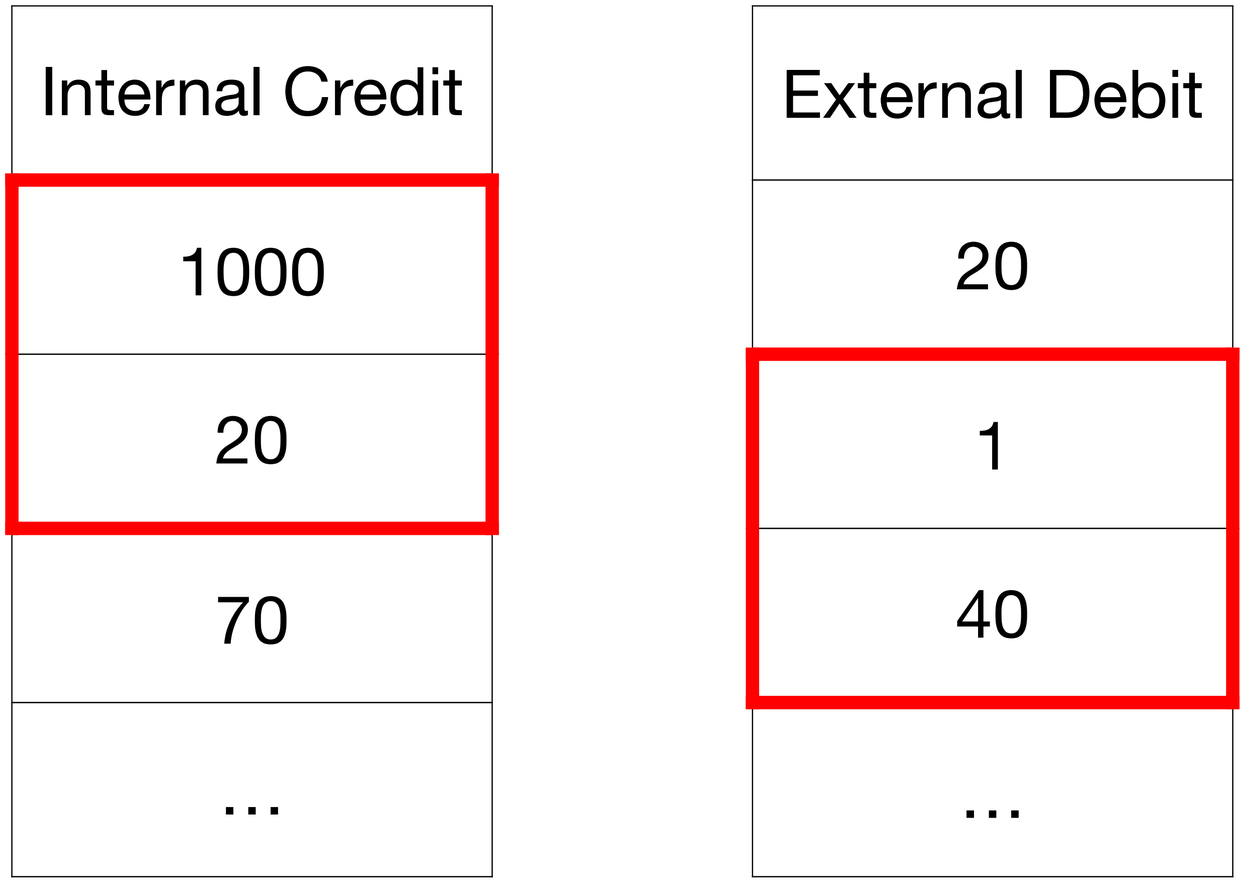
In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square



Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

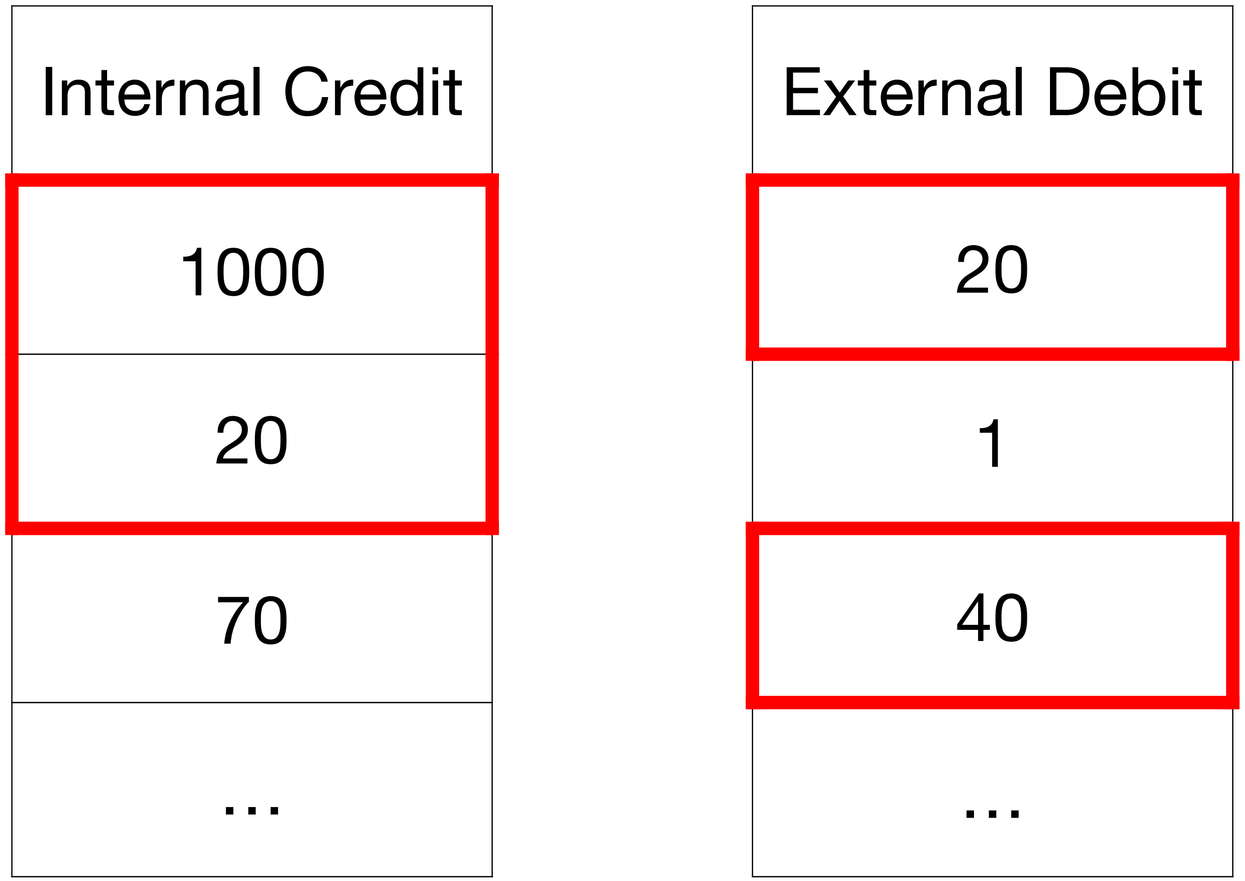
In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square



Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

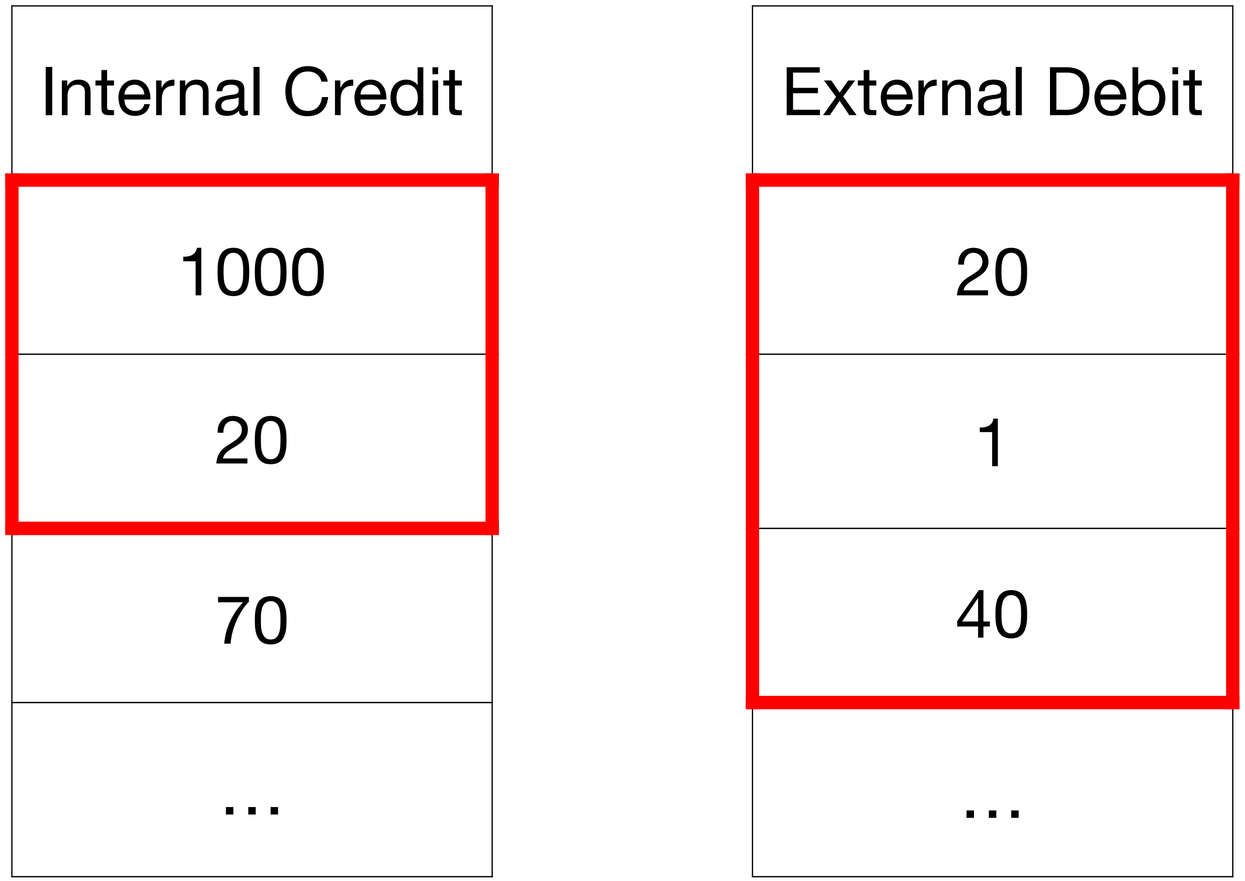
In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square



Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

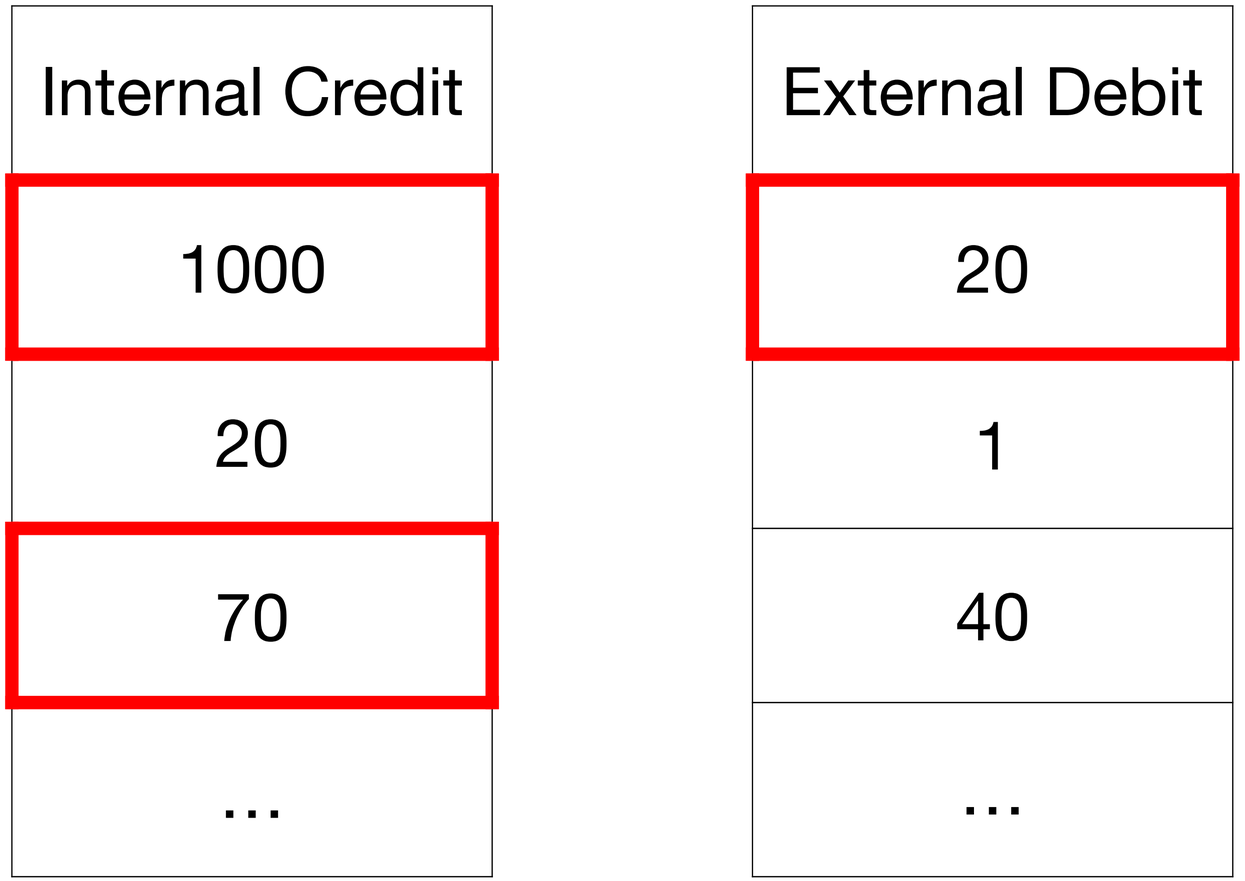
In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square



Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

After layer 1, we only need to handle the remaining data frame.

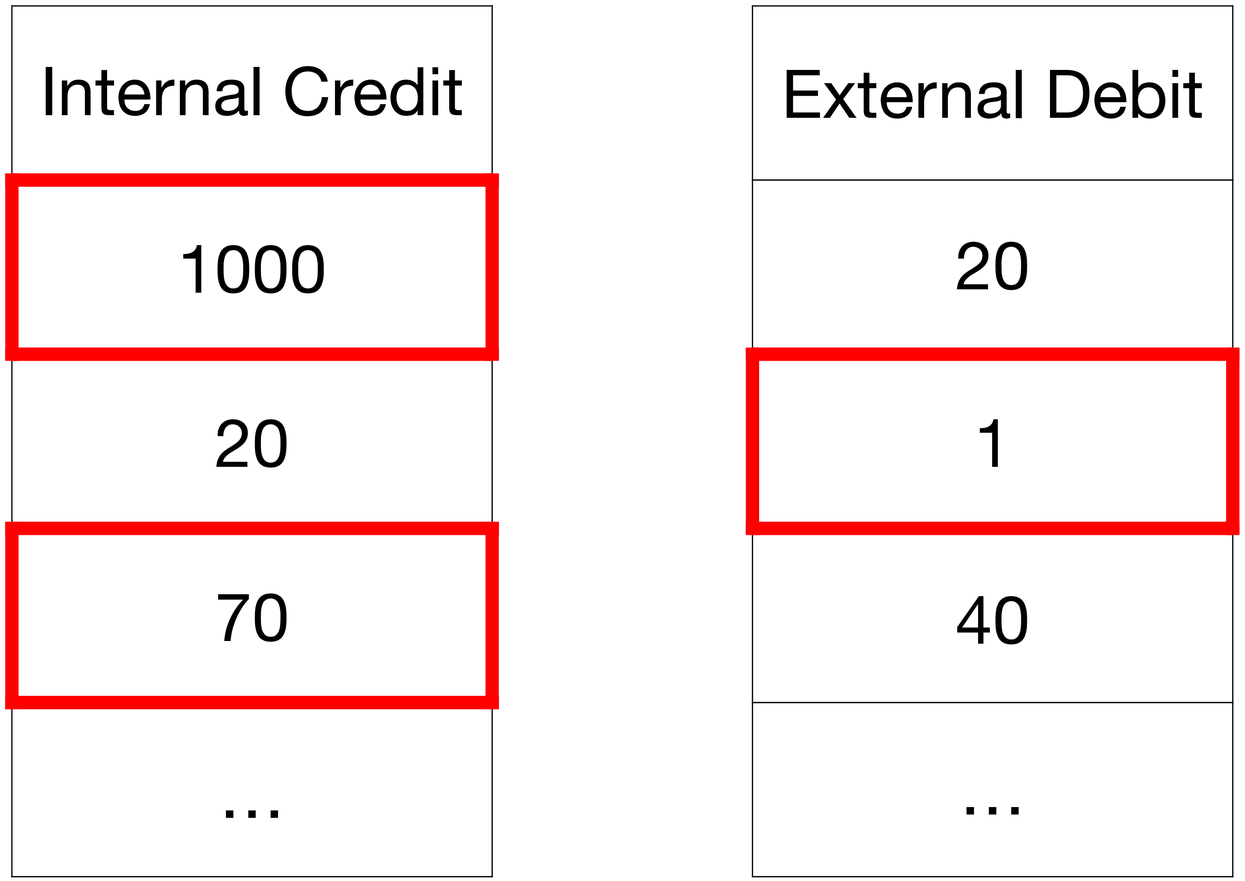
In this layer, combination of the transactions will be calculated; the match combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

Right Red Square - Left Red Square



Assume that today's imbalance for Internal Credit & External Debit is **1030**

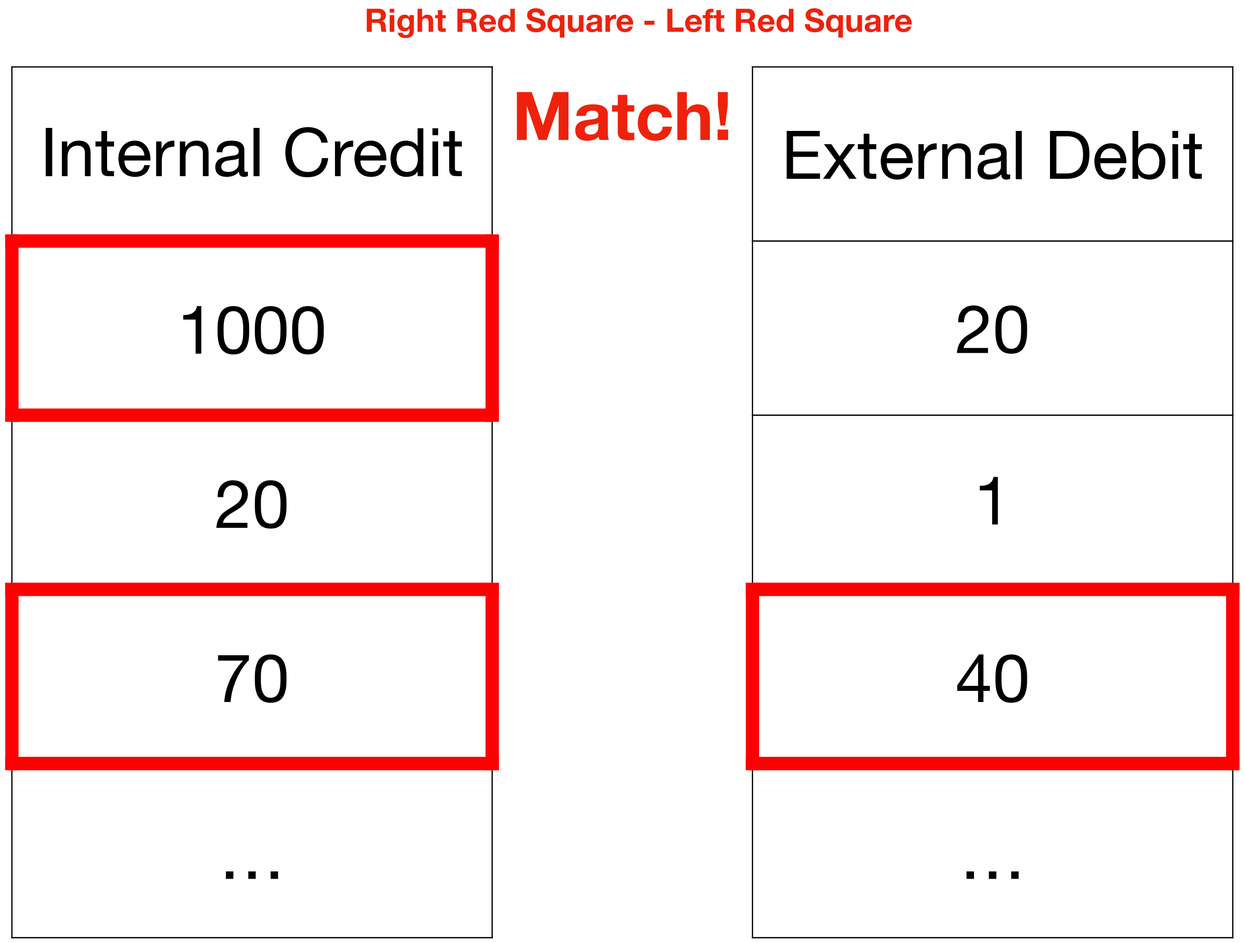
Layer 2: Forward Match

When a combination match, the program will record the instance and continue executing to see if there is a more concise combination. After the completion of iteration, the most concise combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.



Assume that today's imbalance for Internal Credit & External Debit is **1030**

Layer 2: Forward Match

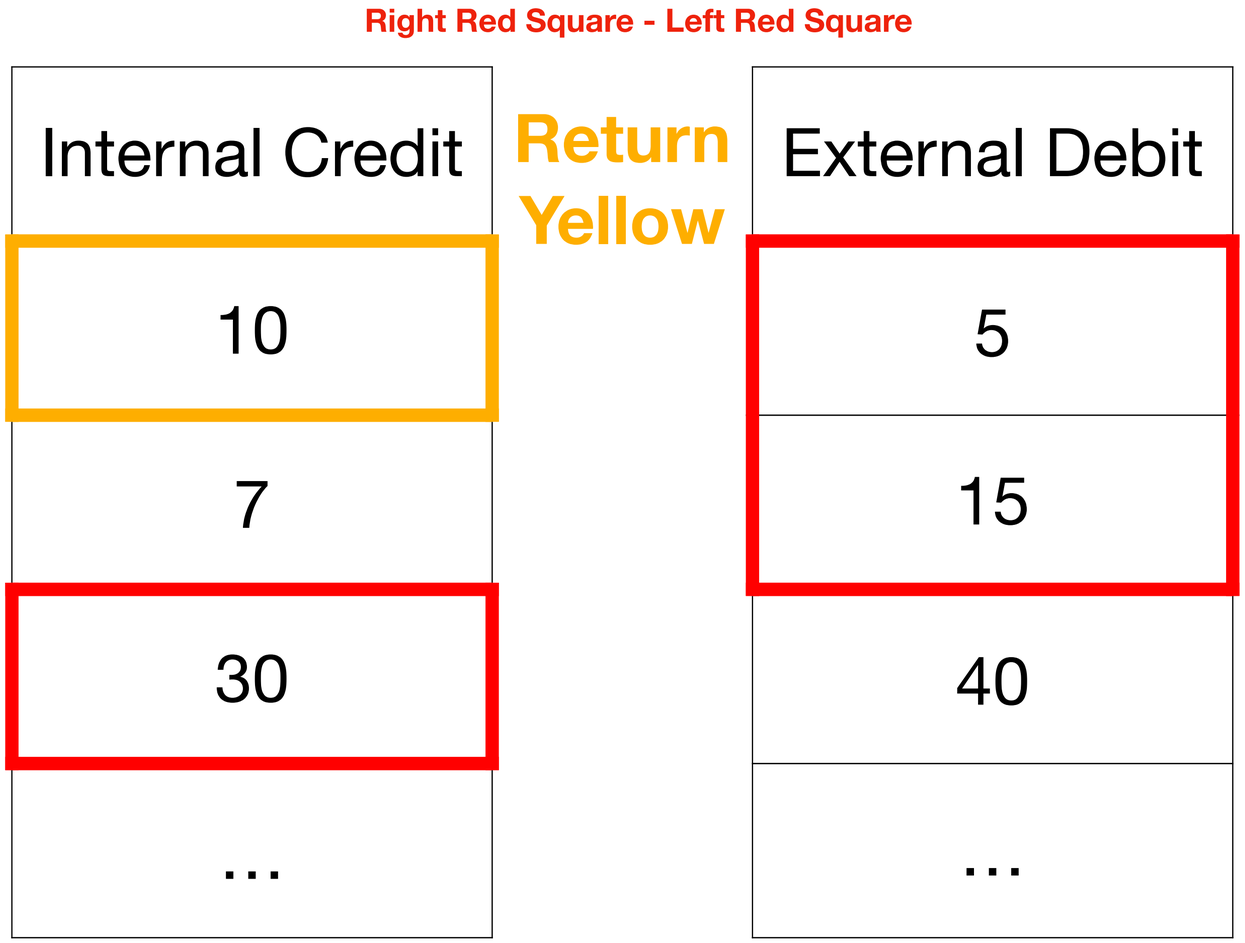
When a combination match, the program will record the instance and continue executing to see if there is a more concise combination. After the completion of iteration, the most concise combination will be returned.

The precedence goes:

comb. with X trans. > comb. with Y trans; where len(X) < len(Y).

(1) comb. with X trans. > (2) comb. with Y trans; where len(X) == len(Y), (i) means the time order of the match.

For example, the yellow square pair will be returned instead of the red square pair.



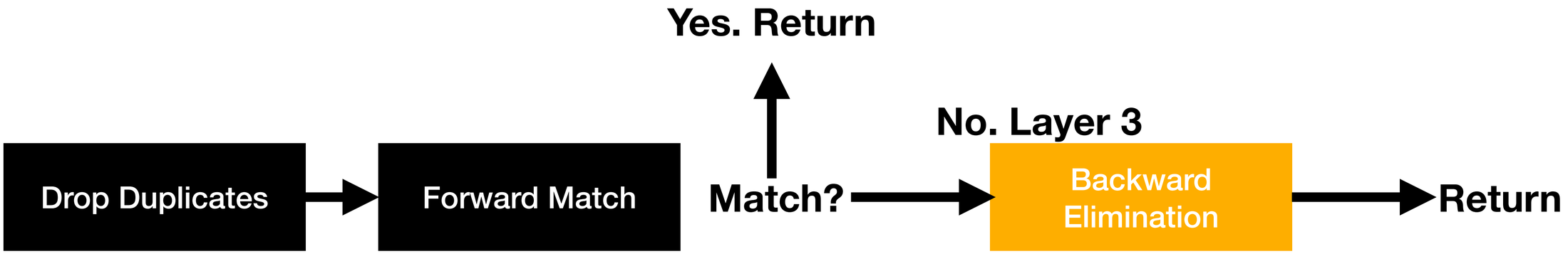
Assume that today's imbalance for Internal Credit & External Debit is **10**

Layer 2: Forward Match; Pseudo Code

```
n_1 = helper_func(internal_account)
n_2 = helper_func(external_account)
internal_combs = get_combination(array=internal_account, choose_to=n_1)
external_combs = get_combination(array=external_account, choose_to=n_2)
Candidates = []

for int_comb in internal_combs:
    for ex_comb in external_combs:
        If internal_account.select(int_comb).sum() - external_account.select(ex_comb).sum() == target:
            candidates.append((int_comb, ex_comb))

shortest = inf
for cand in candidates:
    total_len = len(cand[0]) + len(cand[1)
    If total_len < shortest:
        shortest = total_len
        match_pair = cand
return internal_account.select(cand[0]), external_account.select(cand[1])
```


Yes. Return
No. Layer 3
Drop Duplicates
Forward Match
Match?
Backward Elimination
Return

Layer 3: Backward Elimination

A helper function will be used to determine the optimal selection of the combination of two array.

Currently, the number is set to 1000 per array so that the maximum operation will not exceed 1000 * 1000 + 1000.

Internal Credit
1000
20
70
…

External Debit
20
1
40
…

Layer 2: Forward Match; Pseudo Code

A helper function will be used to determine the optimal selection of the combination of two array.

Currently, the number is set to 1000 per array so that the maximum operation will not exceed 1000 * 1000 + 1000.

The mathematical formula for optimization is:

$$\max(x) \; subject \; to \sum_{i=0}^{x} \binom{n}{t} < 1000 \qquad t = n - i$$

Where x is the optimal number of choose; n is the total length of the input array.

```
def helper_func(array):
    num = len(array)
    counter = 0
    limit = 1000

    for n in num:
      counter += combination(num, n)
      If counter > limit:
        return max(n - 1, 1)
```

Layer 3: Backward Elimination

If the layer 2 cannot find the match transactions, layer 3 will be executed.

In this layer, combination of the transactions will be calculated; the matched combination will be returned.

The precedence will be same as layer 2.

The only difference is that in this layer. The number of choose is started from n, n-1, n-2, etc., where n is the length of the input array.

If the length of the matched array is shorter than the original array. The unmatched parts will be drop and the array will be updated.

Internal Credit
1000
20
70
…

External Debit
20
1
30
…

Assume that today's imbalance for Internal Credit & External Debit is **37.5**

Layer 3: Backward Elimination

If the layer 2 cannot find the match transactions, layer 3 will be executed.

In this layer, combination of the transactions will be calculated; the matched combination will be returned.

The precedence will be same as layer 2.

The only difference is that in this layer. The number of choose is started from n, n-1, n-2, etc., where n is the length of the input array.

If the length of the matched array is shorter than the original array. The unmatched parts will be drop and the array will be updated.

Internal Credit
1000
20
70
…

External Debit
20
1
30
…

Assume that today's imbalance for Internal Credit & External Debit is **37.5**

Layer 3: Backward Elimination

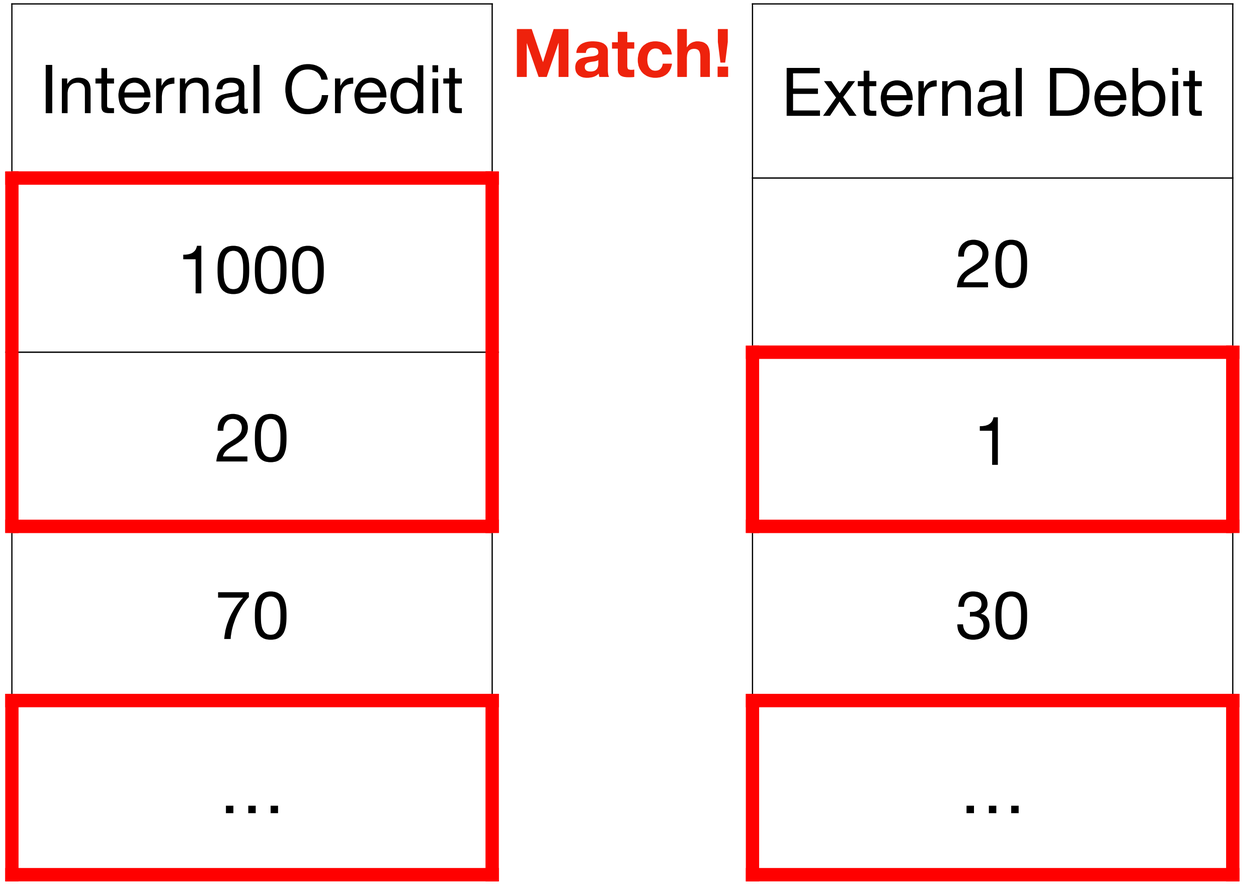
If the layer 2 cannot find the match transactions, layer 3 will be executed.

In this layer, combination of the transactions will be calculated; the matched combination will be returned.

The precedence will be same as layer 2.

The only difference is that in this layer. The number of choose is started from n, n-1, n-2, etc., where n is the length of the input array.

If the length of the matched array is shorter than the original array. The unmatched parts will be drop and the array will be updated.

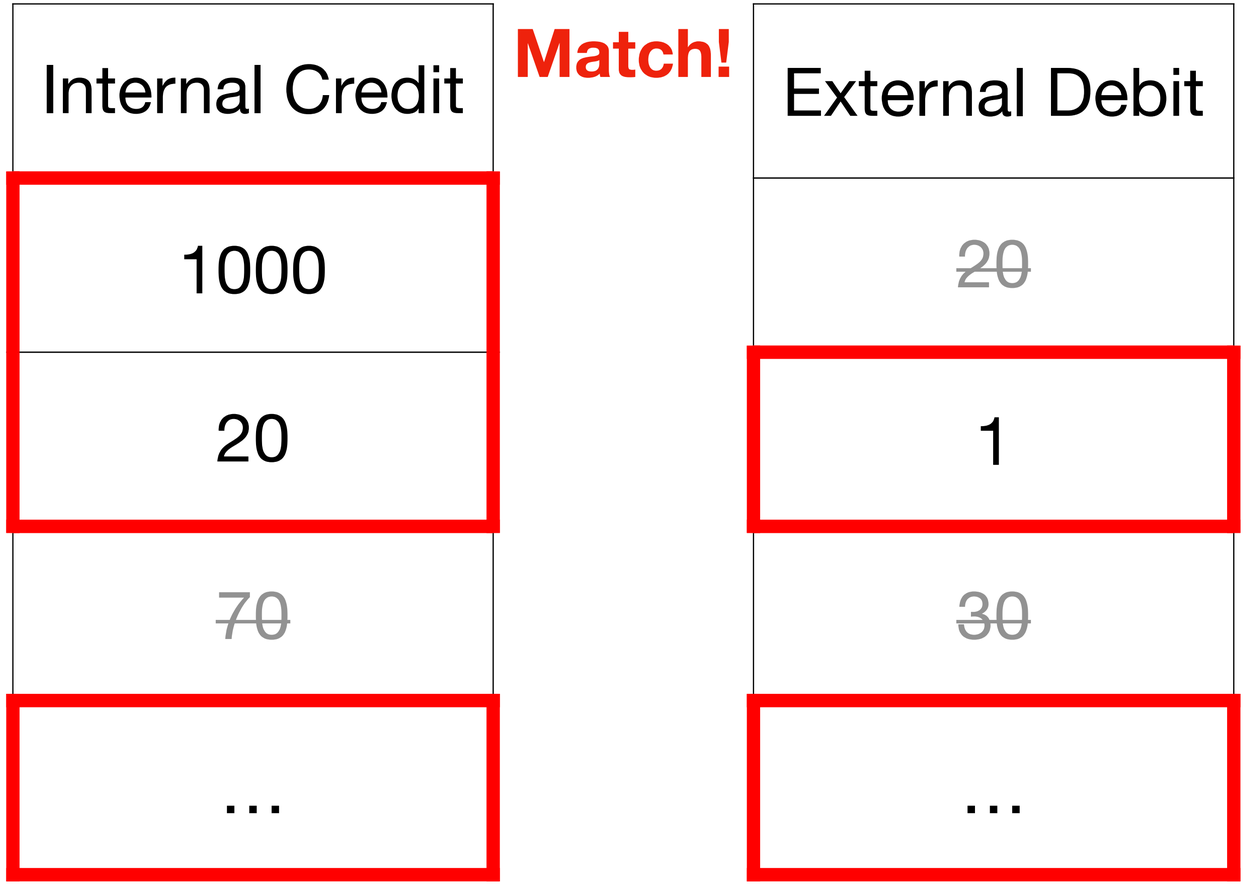


Assume that today's imbalance for Internal Credit & External Debit is **37.5**

Layer 3: Backward Elimination

If the length of the matched array is shorter than the original array, as the example, the unmatched parts will be drop and the array will be updated.

The same operation will be executed in the updated array until there is nothing to be updated (the length of the matched array is equal to the length of the original array)



Assume that today's imbalance for Internal Credit & External Debit is **37.5**

Layer 3: Backward Elimination

If the length of the matched array is shorter than the original array, as the example, the unmatched parts will be drop and the array will be updated.

The same operation will be executed in the updated array until there is nothing to be updated (the length of the matched array is equal to the length of the original array)

Internal Credit	Update!	External Debit
1000		1
20		23
65		78
…		…

Assume that today’s imbalance for Internal Credit & External Debit is **37.5**

Layer 3: Backward Elimination

A helper function will be used to determine the optimal selection of the combination of two array.

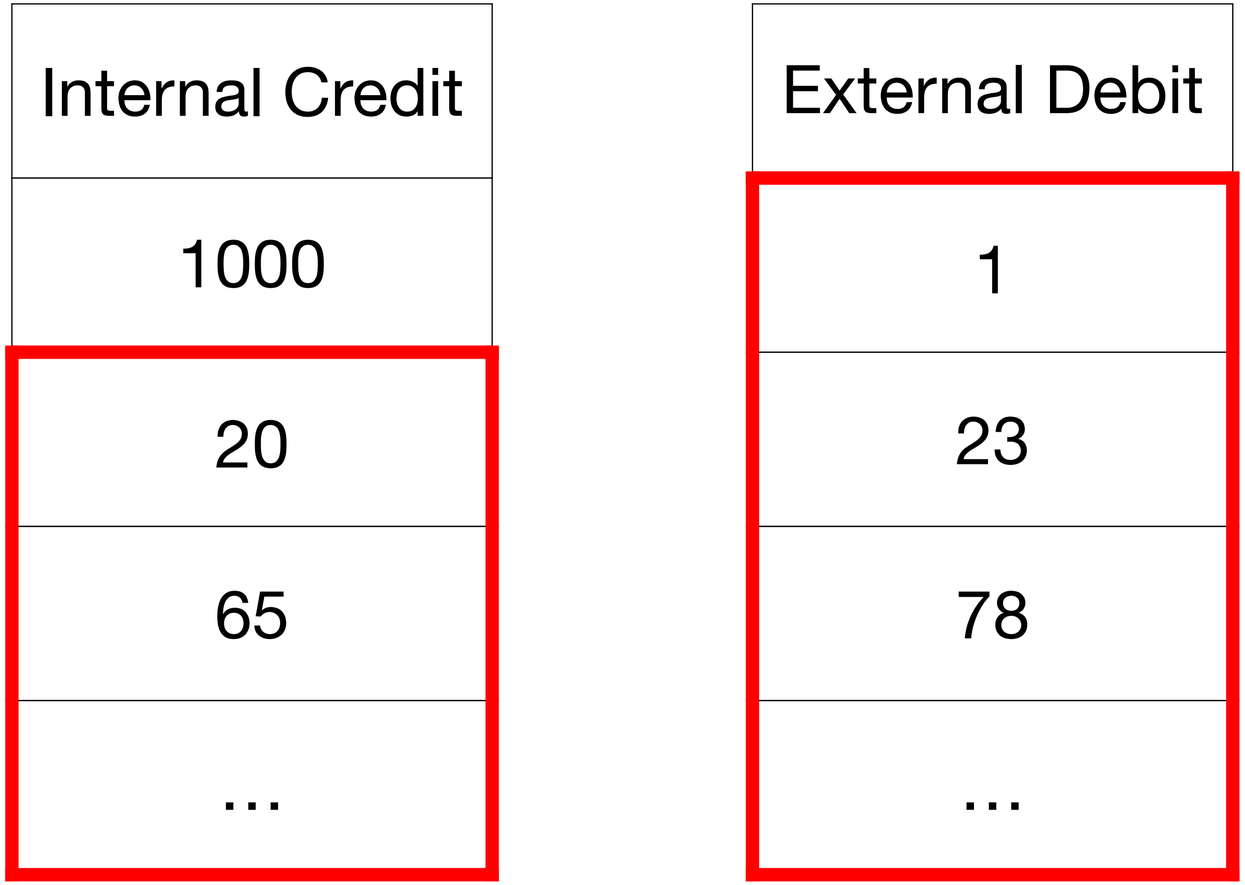
Currently, the number is set to 1000 per array so that the maximum operation will not exceed 1000 * 1000 + 1000.

Internal Credit
1000
20
65
…

External Debit
1
23
78
…

Layer 3: Backward Elimination

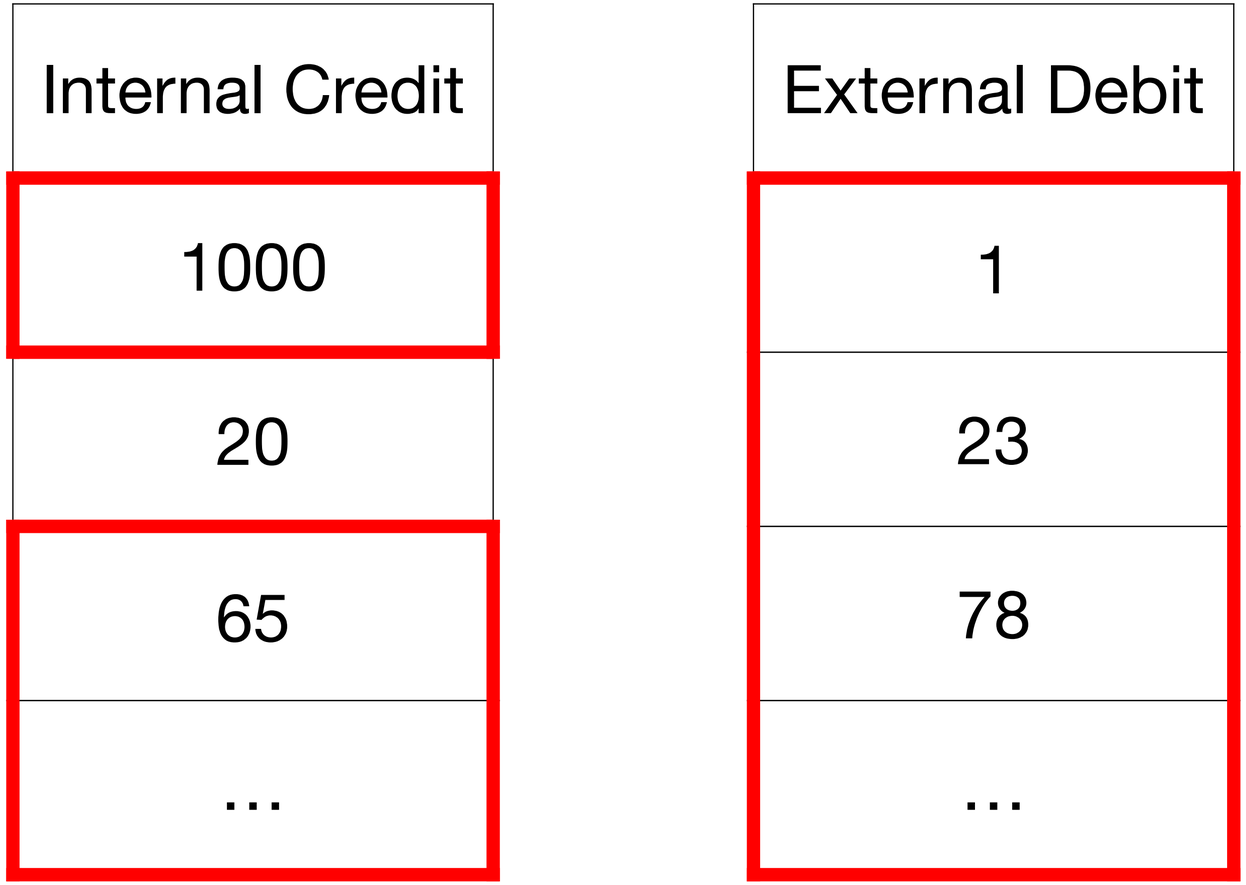
The same operation is executed in the updated array.



Assume that today's imbalance for Internal Credit & External Debit is **37.5**

Layer 3: Backward Elimination

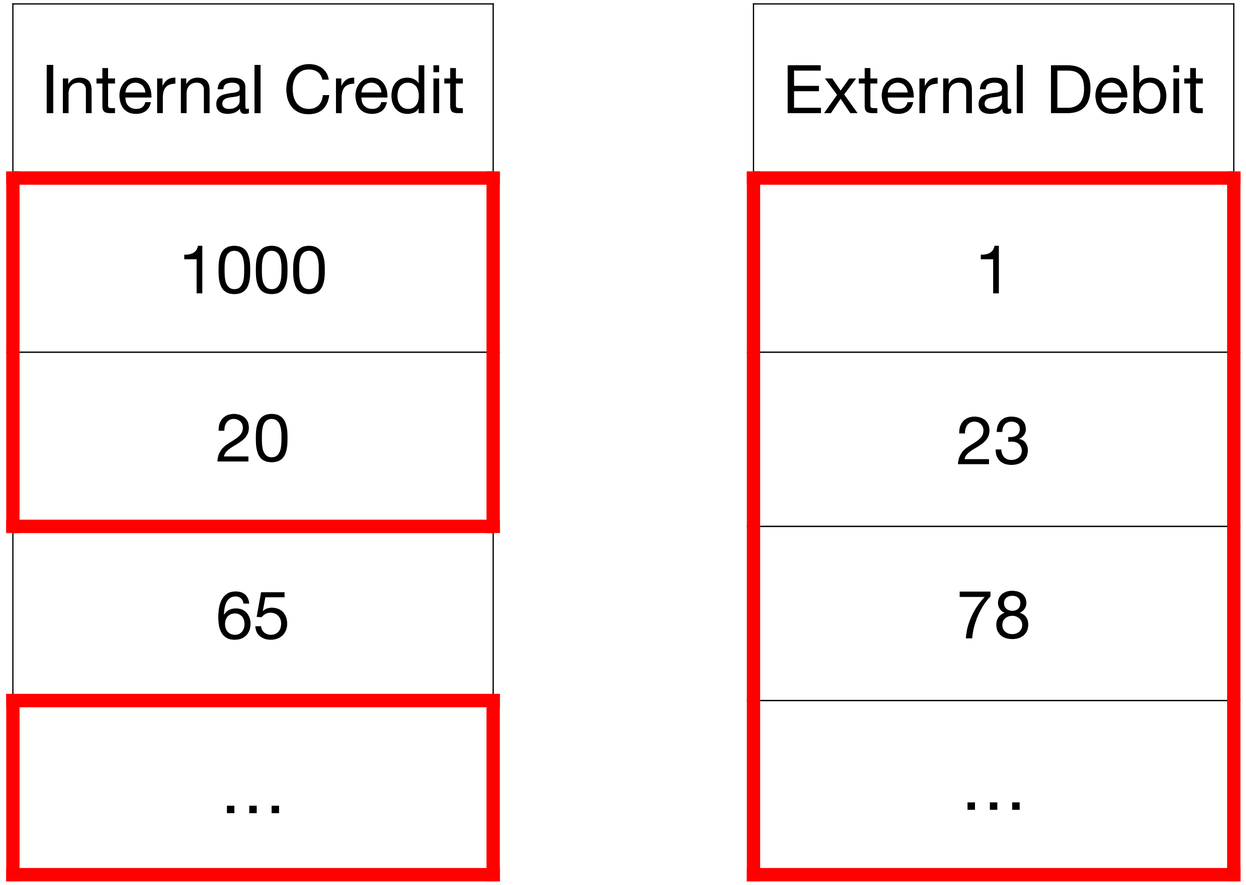
The same operation is executed in the updated array.



Assume that today's imbalance for Internal Credit & External Debit is **37.5**

Layer 3: Backward Elimination

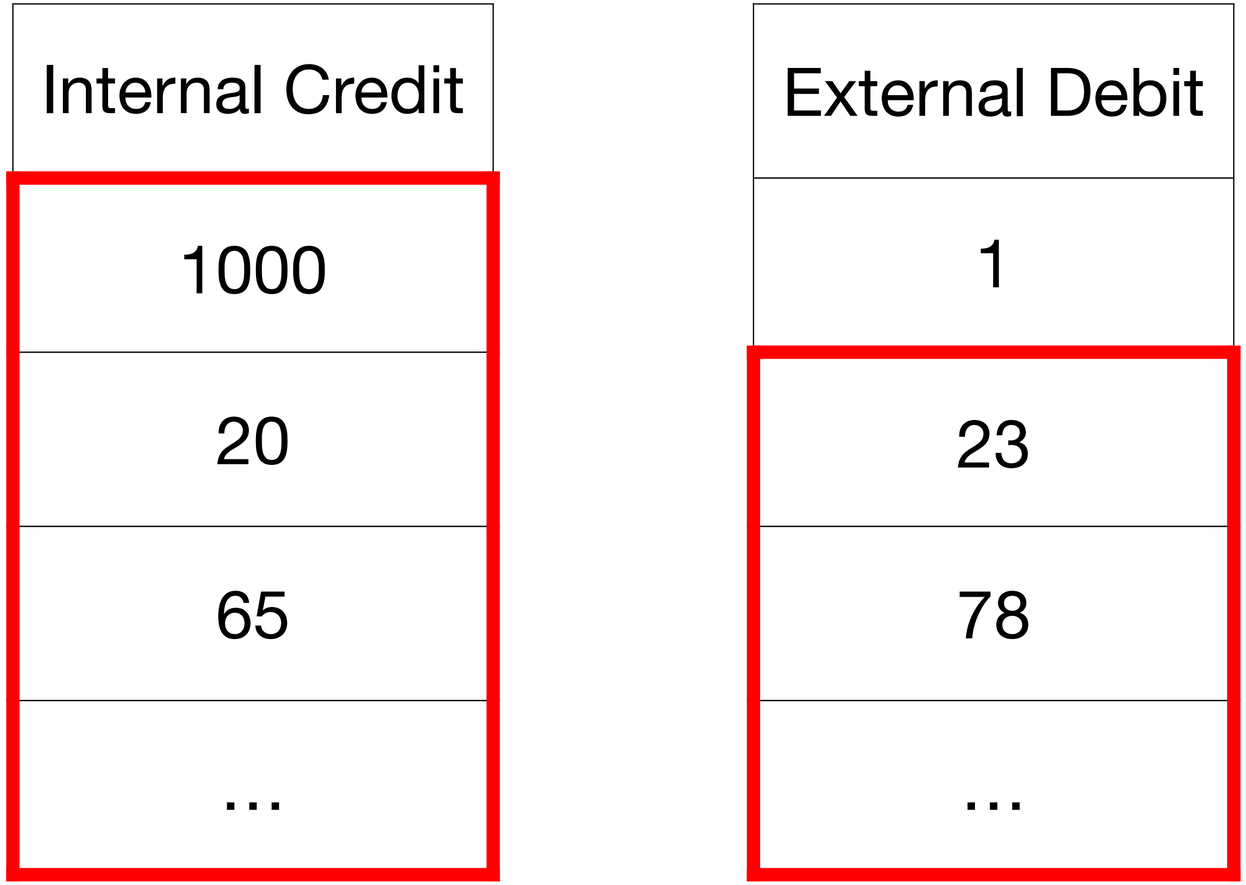
The same operation is executed in the updated array.



Assume that today's imbalance for Internal Credit & External Debit is **37.5**

Layer 3: Backward Elimination

The same operation is executed in the updated array.



Assume that today's imbalance for Internal Credit & External Debit is **37.5**

Layer 3: Backward Elimination

The same operation is executed in the updated array.

Eventually, there is no subset of the original array that can match the imbalance amount under the limitation of the helper function. The entire array will be selected, stop updating, and return the result.

Internal Credit	Match!	External Debit
1000		1
20		23
65		78
…		…

Assume that today’s imbalance for Internal Credit & External Debit is **37.5**

Layer 3: Backward Elimination

The same operation is executed in the updated array.

Eventually, there is no subset of the original array that can match the imbalance amount under the limitation of the helper function. The entire array will be selected, stop updating, and return the result.

Internal Credit	All is Selected Return!	External Debit
1000		1
20		23
65		78
…		…

Assume that today's imbalance for Internal Credit & External Debit is **37.5**

Layer 3: Backward Elimination; Pseudo Code

```
while True:
    n_1 = helper_func(internal_account)
    n_2 = helper_func(external_account)
    internal_combs = get_combination(array=internal_account, choose_to=n_1)
    external_combs = get_combination(array=external_account, choose_to=n_2)
    Candidates = []

    for int_comb in internal_combs:
        for ex_comb in external_combs:
            if internal_account.select(int_comb).sum() - external_account.select(ex_comb).sum() == target:
                candidates.append((int_comb, ex_comb))
```

…to be cont’d in the next page

Layer 3: Backward Elimination; Pseudo Code

```
while True:
    …..cont'd from last page

    shortest = inf
    for cand in candidates:
        total_len = len(cand[0]) + len(cand[1)
        If total_len < shortest:
            shortest = total_len
            match_pair = cand
    condition_1 = len(internal_account) == len(internal_account.select(cand[0]))
    condition_2 = len(external_account) == len(external_account.select(cand[1])

    if condition_1 & condition_2:
        return internal_account, external_account
    else:
        internal_account, external_account = internal_account.select(cand[0]), external_account.select(cand[1])
```