⑧ Splitwise Algorithm Design (C++)

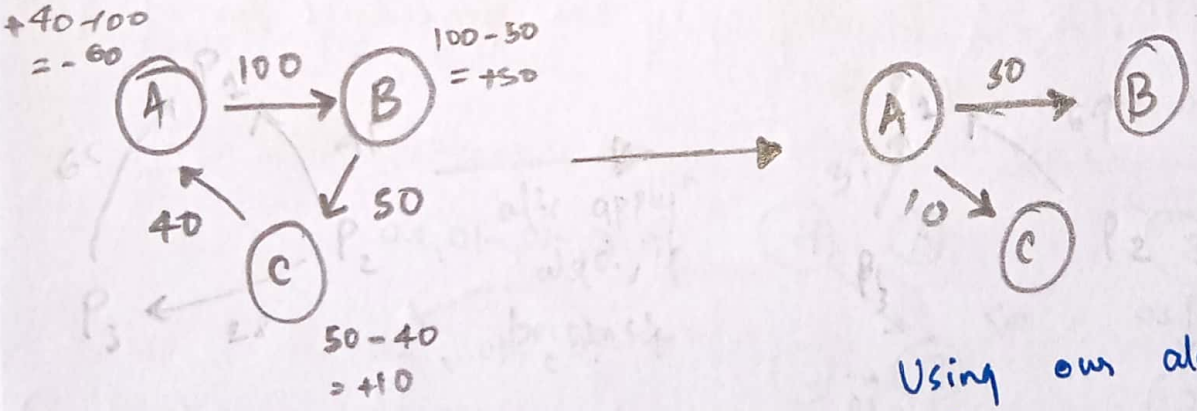
This algo. helps us make money owe network easy. Supose a group of friends buys something together. Now everyone owes each other differnd amts. With many people it becomes difficult to keep track of money etc, so this algo. helps us make things simpler. by minimizing the cash transaction to make things less complicated.

(Cx)

this is the initial network

Debit = $-x$ ₹

Credit = $+x$ ₹



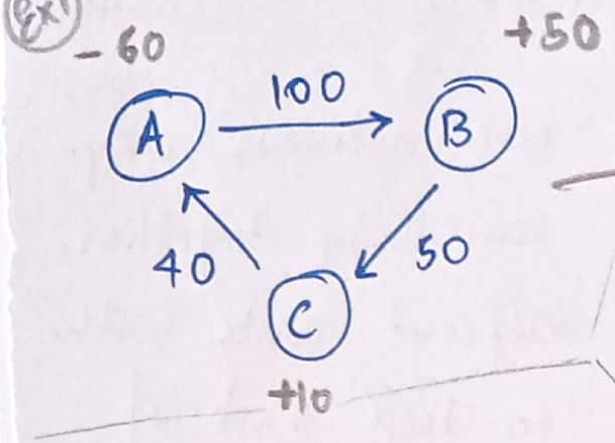
'B' is getth 100₹ & losth 50₹ so net gain for B is 50₹. Similarly C is getth 50 & losth 40 so net gain is 10₹. A is gaining 40₹ & losth 100₹ so net loss for A will be 60₹.

Using our algo we can simplify the transactions! Earlier we required 3 transactns now we only need 2 transactns

Algo

sorted set that allows repeated elmnts

Ex1



what we'll do is add all the net amounts of people in a "multiset" (m)

(who'll pay) debit ⇒ ⇐ credit (ppl who'll get money)

m : -60 , +10 , +50

low high

using organizing the elmnt will take "nlogn" time as each insert operation take 'logn' & if we insrt 'n' elmnts it'll take 'nlogn'.

= -60, 10, 50

bcz min(|-60|, 50) = 50

low = low + 50

high = high - 50

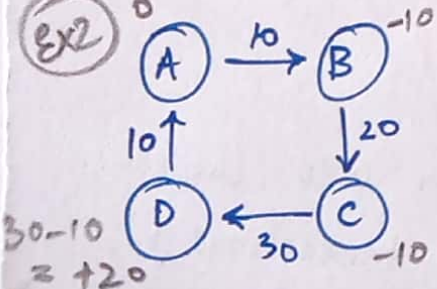
So we'll try to take the smalct no. & try to settle the amount with the largest no. from right.

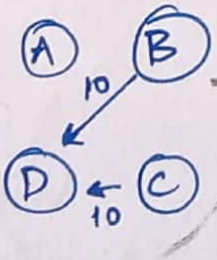
then we'll remove all '0's entries.

⇒ -10, +10, ~~0~~ & finally apply sm procedure for -10 & +10

⇒ ~~0~~, ~~0~~ → 0 ie []

Ex2





= ~~0~~, -10, -10, 20

⇒ -10, -10, 20

⇒ ~~0~~, -10, 10

= min(|-10|, 20) = 10

low high

this means that Σ10 can be settled.

min(|+10|, 10) = -10

= -10, 10

= ~~0~~, ~~0~~

= []

finally when we get empty set we can say all trans. has been handled! & we can also calc. no. of trans, payment details etc!

We'll not exactly create a graph, we'll only use its concept to solve our problem!

```
#include <iostream>
#include <set>

using namespace std;

int main(){
    int no_of_transactions, friends;
    cin >> nt >> f;
    int x, y, amount;
    int net[10000] = {0}
    while (nt--){
        cin >> x >> y >> amount;
        net[x] -= amount;
        net[y] += amount;
    }
```

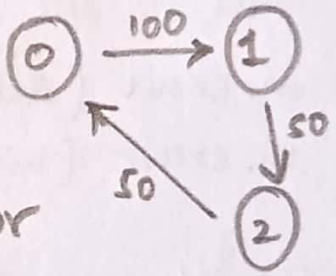
like edges → (nt)

like nodes/vertices → (f)

x —amt→ y

1-D array to store the net amt that each person will have at the end. (debit or credit)

to calc. the net amount for each person.



I/P
3 3
0 1 100
1 2 50
2 0 50

```
// Put a list & sort it => Multiset
multiset<int> m;
for (int i=0; i<friends; i++){
    if (net[i] != 0){
        m.insert(net[i]);
    }
}
```

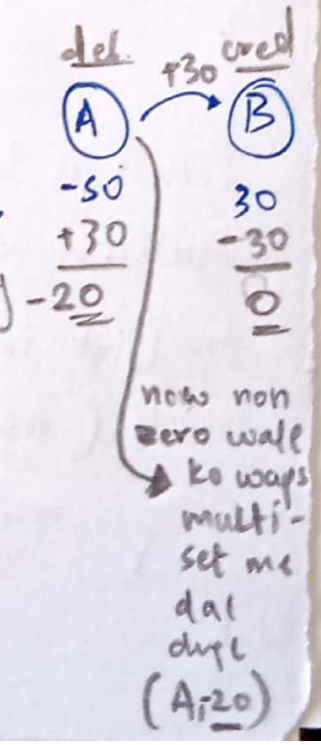
for a person i there exists some transactions

Now we'll take out 2 people. One from left (who'll give money ie debitor) and one from right (who'll get the money ie creditor).

```
int count = 0;          → to count no. of trans.
// pop out 2 ppl (lft & right) & try to settle 'em.
while (!m.empty()) {
    auto left = m.begin();          this gives us a pointr for 1st elem.
    auto right = prev(m.end());     this gus pointr to the last pointr! so for a pointr of last elem we enclose it in prev().
    // to get vals of the pointers
    int debit = *low;
    int credit = *high;

    // Now we'll erase these vals from multiset & afterwards
    // see if the trans is settld or one of 'em is cancelled.
    m.erase (low);
    m.erase (high);

    // Settlement: Now we'll see if trans. is settled or one
                   them cancels out.
    int * settlement_amt = min(-debit, credit)
    count += 1;
    // settlmt amt from debitor to creditor
    debit += settlement_amt;   } (as the settlmt.amt is exactly equal to either debit
    credit -= "      ";        } or credit one of 'em will be equal to 0.)
```



```
if (debit != 0){
    m.insert(debit);
}
if (credit != 0){
    m.insert(credit);
}
}
cout << count;
```

at the end when only 2 val will remain so they'll be of form $-x, +x$. both of them will cancel out & this cond. won't run.

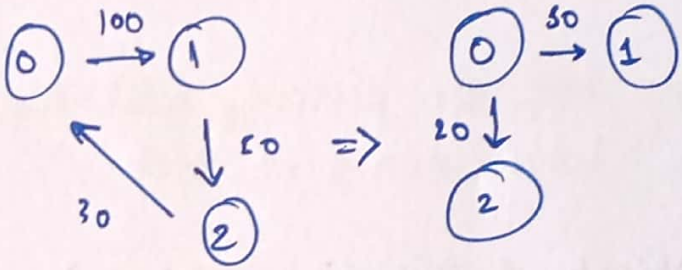
ex

I/P

3 3
0 1 100
1 2 50
2 0 30

O/P

2



~~AA~~ (Main Code)

I/P

3 3
Rahul Ajay 100
Ajay Neha 50
Neha Rahul 40

O/P

2

* for storing names we'll use ~~maps~~ hashmaps

```
int main(){
    int n, t;
    cin >> n >> t;
    string x, y;
    int amount;
    map<string, int> net;
```

(5)

```
while (nt--){
    cin >> x >> y >> amount;
    if (m.count(x) == 0) {
        net[x] = 0;
    }
    if (m.count(y) == 0){
        net[y] = 0;
    }
    net[x] -= amount;
    net[y] += amount;
}

// Iterate over the persons, add those person in the multiset
who have non-zero net

multiset < pair <int, string> > m;

for (auto p: net){
    string person = p.first;
    int amount = p.second;

    if (net[person] != 0){
        m.insert( make_pair(amount, person) );
    }
}
```

```
int count = 0;
while (!m.empty()) {
    auto low = m.begin();
    auto high = m.prev(m.end());
    int debit = low -> first;
    string debit_person = low -> second;
    int credit = high -> first;
    string credit_person = high -> second;
    // pop 'em out.
    m.erase(low);
    m.ersase(high);
    int settlement_amt = min(-debit, credit);
    debit += settlement_amt;
    credit -= settlement_amt;
    // print tran
    cout << debit_per.. << "will pay" << setlm_amt << "to" << cred_pr << endl;
    if (debit != 0) {
        m.insert(make_pair(debit, debit_person));
    }
    if (credit != 0) {
        m.insert(make_pair(credit, cred_pers));
    }
    count += 1;
}
cout << count; }
```

(7)

Q1

3 3

Rahul Ajay 100

Ajay Neha 50

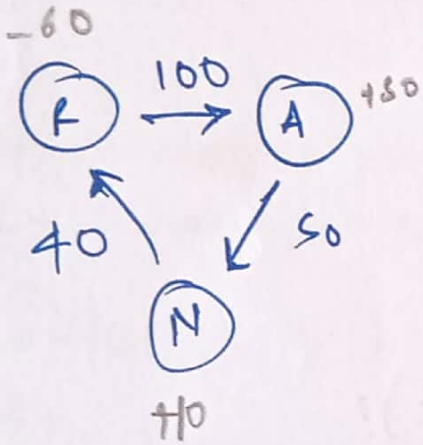
Neha Rahul 40

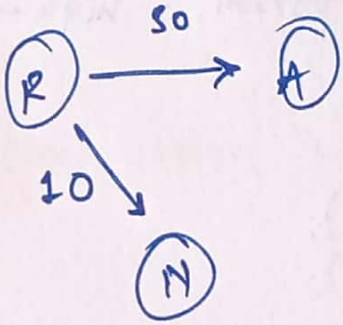
Q2

Rahul will pay 50 to Ajay

Rahul will pay 10 to Neha

2





2 trans.