code warrior

Infrastructure CodeWarrior Challenge #3

Problem #1: Selecting Stocks

Score points: 100

An investor has saved some money and wants to invest in the stock market. There are a number of stocks to choose from, and they wants to buy at most *1* share in any company. The total invested cannot exceed the funds available. A friend who is a stock market expert has predicted the values of each stock after 1 year. Determine the maximum profit that can be earned at the end of the year assuming the predictions come true.

Example

saving = 250

currentValue = [175, 133, 109, 210, 97]

futureValue = [200, 125, 128, 228, 133]

To maximize profits, the investor should buy stocks at indices *2* and *4* for an investment of *109 + 97 = 206*. At the end of the year the stocks are sold for *128 + 133 = 261*, so total profit is *261 - 206 = 55.*

Function Description

Complete the function *selectStock* in the editor below. The function should return an integer that denotes the maximum profit after one year.

selectStock has the following parameter(s):

int *saving:* amount available for investment

int *currentValue[n]:* the current stock values

int *futureValue[n]:* the values of the stocks after one year

Constraints

- $0 < n \le 100$

- 0 < *saving* ≤ 30000
- 0 ≤ *currrentValue[i], futureValue[i]* ≤ 300

Input Format For Custom Testing

The first line contains an integer, *n*, the number of elements in *currrentValue*.
Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer, *currrentValue[i]*.
The next line contains an integer, *n*, the number of elements in *futureValue*.
Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains an integer, *futureValue[i]*.

Sample Case 0

Sample Input For Custom Testing

```
STDIN    Function
-----    --------
30    →  saving = 30
4     →  currentValue[] size n = 4
1     →  currentValue = [1, 2, 4, 6]
2
4
6
4     →  futureValue[] size n = 4
5     →  futureValue = ]5, 3, 5, 6]
3
5
6
```

Sample Output

```
6
```

Explanation

The investor can buy all *4* stocks and gain a profit of *(5-1)+(3-2)+(5-4)+(6-6) = 4+2+1+0 = 6*.

Sample Case 1

Sample Input For Custom Testing

```
STDIN    Function
-----    --------
500   →  saving = 500
5     →  currentValue[] size n = 5
150   →  currentValue = [150, 199, 200, 168, 153]
199
200
168
153
5     →  futureValue[] size n = 5
140   →  futureValue = [140, 175, 199, 121, 111]
175
199
121
111
```

Sample Output

```
0
```

Explanation

All the stocks lose value during the year, so no investment is made. There is no way to make a profit.

Problem #2: Shared Interest
Score points: 100

Given a graph of friends who have different interests, determine which groups of friends have the most interests in common. Then use a little math to determine a value to return.

The graph will be represented as a series of nodes numbered consecutively from *1* to *friends_nodes*. Friendships have evolved based on interests which will be represented as weights in the graph. Any members who share the same interest are said to be connected by that interest. Once the node pairs with the maximum number of shared interests are determined, multiply the friends_nodes of the resulting node pairs and return the maximal product.

Example

friends_nodes = 4

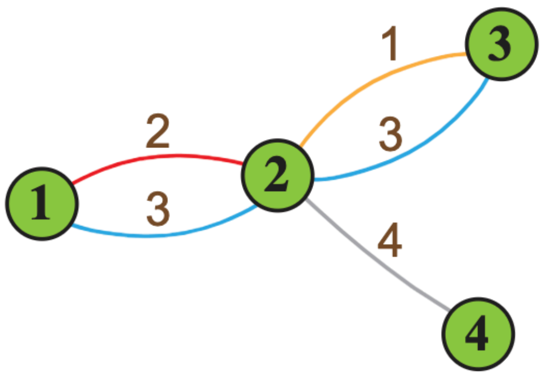
friends_edges = 5

friends_from = [1, 1, 2, 2, 2]

friends_to = [2, 2, 3, 3, 4]

friends_weight = [2, 3, 1, 3, 4]

```
From  To    Weight
1     2     2
1     2     3
2     3     1
2     3     3
2     4     4
```



The graph shows the following connections:

```
 Weight
(Interest)    Connectons
    1             2,3
    2             1,2
    3             1,2,3
    4             2,4
```

- Node pair (2,4) shares only 1 interest (4) and node pair (1,3) shares 1 interest (3).

- Node pair (1,2) shares 2 interests (interests 2 and 3) and node pair (2, 3) shares also 2 interests (interest 1 and 3) . So, the maximum number of shared interests is 2.
- Multiply the friends_nodes of the resulting node pairs : *1 × 2 = 2* and *2 × 3 = 6*.
- The maximal product is 6.

Function Description

Complete the function *maxShared* in the editor below.

maxShared has the following parameter(s):

int *friends_nodes:* number of nodes

int *friends_from[friends_edges]:* the first part of node pairs

int *friends_to[friends_edges]:* the other part of node pairs

int *friends_weight[friends_edges]:* the interests of node pairs

Returns:

int: maximal integer product of all node pairs sharing the most interests.

Constraints

- $2 \le friends\_nodes \le 100$
- $1 \le friends\_edges \le \min(200, {}^{(friends\_nodes \times (friends\_nodes - 1))} / {}_{2})$
- $1 \le friends\_weight[i] \le 100$
- $1 \le friends\_from[i], friends\_to[i] \le friends\_nodes$
- $1 \le friends\_weight[i] \le friends\_edges$
- $friends\_from[i] \ne friends\_to[i]$
- Each pair of friends can be connected by zero or more interests.

Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

The first line contains two space-separated integers *friends_nodes* and *friends_edges*.

Each of the next *friends_edges* lines contains three space-separated integers, *friends_from[i]*, *friends_to[i]* and *friends_weight[i]* where 0 ≤ i < *friends_edges*.

Sample Case 0

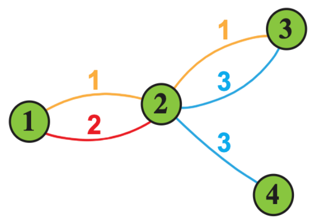
Sample Input 0

```
STDIN     Function
-----     --------
4 5     →  friends_nodes = 4  friends_edges = 5
1 2 1   →  friends_from = [1,1,2,2,2]  friends_to = [2,2,3,3,4]  friends_weight = [1,2,1,3,3]
1 2 2
2 3 1
2 3 3
2 4 3
```

Sample Output 0

```
6
```

Explanation 0

Each pair of *friends_nodes = 4* friends is connected by the following interests: • Pair *(1, 2)* shares *2* interests (i.e., interests *1* and *2*) • Pair *(1, 3)* shares *1* interest (i.e., interest *1*) • Pair *(1, 4)* shares *0* interests • Pair *(2, 3)* shares *2* interests (i.e., interests *1* and *3*) • Pair *(2, 4)* shares *1* interest (i.e., interest *3*) • Pair *(3, 4)* shares *1* interest (i.e., interest *3*)	 The pairs connected by the maximal number of interests are *(1, 2)* and *(2, 3)*. Their respective products are *1 × 2 = 2* and *2 × 3 = 6*. The result is the the largest of these values which is *6*.