Mako
2021 May 01
score 100

Q1. Two sum variant
This is a two-sum variant.

Write a function:

```
int solution(vector<int> &A);
```

that, given an array A consisting of N integers, returns the maximum sum of two numbers whose digits add up to an equal sum. If there are no two numbers whose digits have an equal sum, the function should return -1.

Examples:

1. Given A = [51, 71, 17, 42], the function should return 93. There are two pairs of numbers whose digits add up to an equal sum: (51, 42) and (17, 71). The first pair sums up to 93.

2. Given A = [42, 33, 60], the function should return 102. The digits of all numbers in A add up to the same sum, and choosing to add 42 and 60 gives the result 102.

3. Given A = [51, 32, 43], the function should return -1, since all numbers in A have digits that add up to different, unique sums.

Write an **efficient** algorithm for the following assumptions:

- N is an integer within the range [1..200,000];
- each element of array A is an integer within the range [1..1,000,000,000].

Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.

Solution
in linear time

```cpp
int solution(vector<int> &A)
{
    std::vector<int> B;
    for(const auto& x:A)
    {
        B.push_back(sum_of_digits(x));
    }

    int ans = -1;
    std::unordered_map<int, int> hist;
    for(std::uint32_t n=0; n!=A.size(); ++n)
    {
        auto iter = hist.find(B[n]);
        if (iter != hist.end())
        {
            auto temp = A[iter->second] + A[n];
            if (ans ==-1 || ans < temp) ans = temp;
            if (A[iter->second] < A[n]) iter->second = n;
        }
        else
        {
            hist[B[n]] = n;
        }
    }
    return ans;
}
```

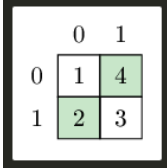
`hist[i] = maximum(A[0], A[1], ..., A[n], s.t. digit_sum(A[n]==i)`

Q2. Two sum on a chessboard

This is a two-sum variant. With special maths property, do it in *O(MN)* time ...

You are given a matrix A representing a chessboard with N rows and M columns. Each square of the chessboard contains an integer representing a points-based score. You have to place two rooks on the chessboard in such a way that they cannot attack each other and the sum of points on their squares is maximal. Rooks in chess can attack each other only if they are in the same row or column.

For example, given a matrix as in the following picture:



	0	1
0	1	4
1	2	3

we can place rooks in two different ways:

- One rook on A[0][0] = 1 and another rook on A[1][1] = 3. The sum of points on these squares is 4.
- One rook on A[0][1] = 4 and another rook on A[1][0] = 2. The sum of points on these squares is 6.

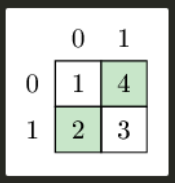
Your task is to find the maximum sum of two squares of the chessboards on which the rooks can be placed. In the example above, the answer is 6. We cannot, for example, place the rooks at A[0][1] and A[1][1] (whose sum is 7), as they would attack each other.

Write a function:

```
int solution(vector< vector<int> > &A);
```

which, given a matrix A, returns the maximum sum that can be achieved by placing two non-attacking rooks.

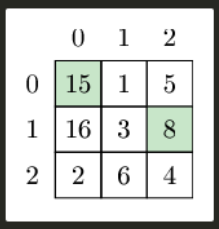
1. Given matrix A with two rows and two columns:



	0	1
0	1	4
1	2	3

the function should return 6. We can achieve the maximum sum by selecting A[0][1] + A[1][0] = 4 + 2. The selected squares are marked in green.

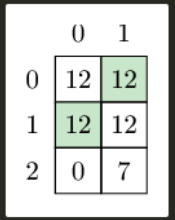
2. Given matrix A with three rows and three columns:



	0	1	2
0	15	1	5
1	16	3	8
2	2	6	4

the function should return 23. We can achieve the maximum sum by selecting A[0][0] + A[1][2] + 8. The selected squares are marked in green.

3. Given matrix A with three rows and two columns:



	0	1
0	12	12
1	12	12
2	0	7

the function should return 24. We can achieve the maximum sum by selecting A[0][0] + A[1][0] + 12 or A[0][1] + A[1][0] = 12 + 12. The latter solution is marked in green.

4. Given matrix A with two rows and three columns:

	0	1	2
0	1	2	14
1	8	3	15

the function should return 22. We can achieve the maximum sum by selecting A[0][2] + A[1][0] = 14 + 8. The selected squares are marked in green.

Write an **efficient** algorithm for the following assumptions:

- N and M are integers within the range [2..600];
- each element of matrix A is an integer within the range [0..1,000,000,000].

Solution

No dynamic programming is needed. Instead make use of the property that, both selected numbers should either be :

- the largest or second largest in its row, and at the same time
- the largest or second largest in its column

Solution from stackoverflow

1. For each row, find the top 2 values and remember the column where they were found. *O(mn)*
2. For each column, find the top 2 values and remember the row where they were found. *O(mn)*
3. The the remaining operations, we only use the two lists built above. We will not look at the matrix again:
    1. For each row, pretend to place a rook in that row and in the column with the highest value. For each column, sum that top value with the top value for the column, except for the column where the rook is, where we sum with the second highest value. Remember the row of the pretend rook and the column with the highest sum. *O(mn)*
    2. Repeat, but use the second highest value. *O(mn)*

Operation complete. *O(mn)* + *O(mn)* + *O(mn)* + *O(mn)* = ***O(mn)***