**Text 1(15)**

**Topic description：**

I have a very simple problem for you. Given two integers A and B, your job is to calculate the Sum of A + B.

**Requirement**

time limitation：1000ms, memory limitation：100MB

**Input**

The first line of the input contains an integer T(1≤T≤20) which means the number of test cases.

Then T lines follow, each line consists of two positive integers, A and B. Notice that the integers are very large,that means you should not process them by using 32-bit integer.You may assume the length of each integer will not exceed 1000.

**Output**

For each test case, you should output two lines. The first line is "Case #:", # means the number of the test case. The second line is the an equation "A + B = Sum", Sum means the result of A + B.Note there are some spaces int the equation. Output a blank line between two test cases.

**Example 1**

Input

2

1 2

112233445566778899 998877665544332211

Output

Case 1:

1 + 2 = 3

Case 2:

112233445566778899 + 998877665544332211 = 1111111111111111110

**Function**

#include <iostream>

#include <vector>

#include <string>

std::string reverse(std::string const &s)

{

std::string rev;

std::string::const\_reverse\_iterator it = s.crbegin();

while (it != s.crend()) {

rev = rev.append(1, \*(it++));

};

return rev;

}

void cinValues(std::string \*s\_a, std::string \*s\_b, int n) {

for (int i = 0; i < n; ++i) {

std::cin >> s\_a[i] >> s\_b[i];

s\_a[i] = reverse(s\_a[i]);

s\_b[i] = reverse(s\_b[i]);

}

}

void coutValues(std::string \*s\_a, std::string \*s\_b, std::vector<int> \*sum, int n) {

int euclidean = 0;

size\_t size\_a;

size\_t size\_b;

size\_t min\_size;

size\_t current\_id = 0;

for (int i = 0; i < n; ++i) {

size\_a = s\_a[i].size();

size\_b = s\_b[i].size();

min\_size = std::min(size\_a, size\_b);

for (int j = 0; j < min\_size; ++j) {

euclidean = euclidean +((int)(s\_a[i].at(j)) - 48) + ((int)(s\_b[i].at(j)) - 48);

sum[i].push\_back(euclidean % 10);

euclidean /= 10;

current\_id = j;

}

current\_id++;

if(size\_a > min\_size) {

for (int j = current\_id; j < size\_a; ++j) {

euclidean = euclidean + ((int)(s\_a[i].at(j)) - 48);

sum[i].push\_back(euclidean % 10);

euclidean /= 10;

}

if (euclidean % 10 > 0) sum[i].push\_back(euclidean % 10);

}

else if(size\_b > min\_size) {

for (int j = current\_id; j < size\_b; ++j) {

euclidean = euclidean + ((int)(s\_b[i].at(j)) - 48);

sum[i].push\_back(euclidean % 10);

euclidean /= 10;

}

if (euclidean % 10 > 0) sum[i].push\_back(euclidean % 10);

}

else {

if (euclidean % 10 > 0) sum[i].push\_back(euclidean % 10);

}

current\_id = 0;

euclidean = 0;

}

for (int i = 0; i < n; ++i) {

std::cout << "Case " << i + 1 << ":" << std::endl;

std::cout << reverse(s\_a[i]);

std::cout << " + ";

std::cout << reverse(s\_b[i]);

std::cout << " = ";

for (auto it = sum[i].crbegin(); it != sum[i].crend(); it++) {

std::cout << \*it;

}

std::cout << std::endl;

std::cout << std::endl;

}

}

int main() {

unsigned int n = 0;

std::cin >> n;

if(n <= 0 || n > 20) return -1;

std::string a[n];

std::string b[n];

std::vector<int> sum[n];

cinValues(a, b, n);

coutValues(a, b, sum, n);

return 0;

}

**Text 2(58)**

Saya likes math, because she thinks math can make her cleverer.

One day, Kudo invited a very simple game:

Given N integers, then the players choose no more than four integers from them (can be repeated) and add them together. Finally, the one whose sum is the largest wins the game. It seems very simple, but there is one more condition: the sum shouldn’t larger than a number M.

Saya is very interest in this game. She says that since the number of integers is finite, we can enumerate all the selecting and find the largest sum. Saya calls the largest sum Greatest Number (GN). After reflecting for a while, Saya declares that she found the GN and shows her answer.

Kudo wants to know whether Saya’s answer is the best, so she comes to you for help.

Can you help her to compute the GN?

**Requirement**

time limitation：1000ms, memory limitation：100MB

**Input**

The input consists of several test cases.

The first line of input in each test case contains two integers N (0<N≤1000) and M(0<M≤ 1000000000), which represent the number of integers and the upper bound.

Each of the next N lines contains the integers. (Not larger than 1000000000)

The last case is followed by a line containing two zeros.

**Output**

For each case, print the case number (1, 2 …) and the GN.

Your output format should imitate the sample output. Print a blank line between each test case.

**Example**

Input

2 10

100

2

0 0

**Output**

Case 1: 8

**Function**

#include <iostream>

#include <set>

#include <vector>

int main(){

std::set<int> sums;

std::vector<int> vect;

int n,m;

int cases = 0;

int input;

for(;;) {

std::cin >> n >> m;

if(n==0 && m==0) break;

for(int i = 0; i < n; i++) {

std::cin >> input;

if(input <= m) {

sums.insert(input);

}

}

for(auto i : sums) {

if(i + i + i + i <= m) vect.push\_back(i + i + i + i);

else if(i + i + i <= m) vect.push\_back(i + i + i);

else if(i + i <= m) vect.push\_back(i + i);

for(auto b : sums) {

if(i + b + b + b <= m) vect.push\_back(i + b + b + b);

if(i + i + b + b <= m) vect.push\_back(i + i + b + b);

if(i + i + i + b <= m) vect.push\_back(i + i + i + b);

}

}

for (const int &i : vect) {

sums.insert(i);

}

cases++;

std::cout << "Case " << cases <<": " << \*(sums.rbegin()) << std::endl;

std::cout << std::endl;

sums.clear();

vect.clear();

}

return 0;

}

**Test 3(74)**

Your boss once had got many copies of a treasure map. Unfortunately, all the copies are now broken to many rectangular pieces, and what make it worse, he has lost some of the pieces. Luckily, it is possible to figure out the position of each piece in the original map. Now the boss asks you, the talent programmer, to make a complete treasure map with these pieces. You need to make only one complete map and it is not necessary to use all the pieces. But remember, pieces are not allowed to overlap with each other (See sample 2).

**Requirement**

time limitation：3000ms, memory limitation：100MB

**Input**

The first line of the input contains an integer T (T <= 10), indicating the number of cases.

For each case, the first line contains three integers n m p (1 <= n, m <= 30, 1 <= p <= 100), the width and the height of the map, and the number of pieces. Then p lines follow, each consists of four integers x1 y1 x2 y2 (0 <= x1 < x2 <= n, 0 <= y1 < y2 <= m), where (x1, y1) is the coordinate of the lower-left corner of the rectangular piece, and (x2, y2) is the coordinate of the upper-right corner in the original map.

Cases are separated by one blank line.

**Output**

If you can make a complete map with these pieces, output the least number of pieces you need to achieve this. If it is impossible to make one complete map, just output -1.

**Example**

Input

3

5 5 1

0 0 5 5

5 5 2

0 0 3 5

2 0 5 5

30 30 5

0 0 30 10

0 10 30 20

0 20 30 30

0 0 15 30

15 0 30 30

Output

1

-1

2

**Function**

#include <iostream>

#include <algorithm>

#include <vector>

#define SIZE 1000000

struct algorithm\_x

{

private:

std::vector<int> L;

std::vector<int> R;

std::vector<int> U;

std::vector<int> D;

std::vector<int> col;

std::vector<int> row;

std::vector<int> H;

std::vector<int> S;

int answer;

int \_n, \_m, \_len;

public:

void init(int n,int m) {

L.resize(SIZE);

R.resize(SIZE);

U.resize(SIZE);

D.resize(SIZE);

col.resize(SIZE);

row.resize(SIZE);

H.resize(SIZE);

S.resize(SIZE);

\_n = n;

\_m = m;

for(int i = 0; i <= \_m; i++) {

U[i] = i;

D[i] = i;

L[i] = i - 1;

R[i] = i + 1;

S[i] = 0;

}

R[m] = 0;

L[0] = \_m;

\_len = \_m;

answer = -1;

for(int i = 1; i <= n; i++) H[i] = -1;

}

int getAnswer() {

return answer;

}

void link(int r,int j) {

D[++\_len] = D[j];

U[D[j]] = \_len;

U[\_len] = j;

D[j] = \_len;

row[\_len] = r;

col[\_len] = j;

S[j]++;

if(H[r] < 0) {

H[r] = \_len;

R[\_len] = \_len;

L[\_len] =\_len;

}

else {

R[\_len] = R[H[r]];

L[R[H[r]]] = \_len;

L[\_len] = H[r];

R[H[r]] = \_len;

}

}

void remove(int c) {

L[R[c]]=L[c];

R[L[c]]=R[c];

for(int i=D[c];i!=c;i=D[i]) {

for(int j=R[i];j!=i;j=R[j]) {

U[D[j]]=U[j];

D[U[j]]=D[j];

S[col[j]]--;

}

}

}

void resume(int r) {

for(int i = U[r]; i != r; i = U[i]) {

for(int j = L[i]; j != i; j = L[j]) {

U[D[j]] = j;

D[U[j]] = j;

S[col[j]]++;

}

}

L[R[r]] = r;

R[L[r]] = r;

}

void dance(int d)

{

if(answer != -1 && d >= answer) return;

if(R[0] == 0) {

if(answer == -1) answer = d;

else answer = std::min(answer, d);

return;

}

int c = R[0];

for(int i = R[0]; i != 0; i = R[i]) {

if(S[i] < S[c]) c = i;

}

remove(c);

for(int i = D[c]; i != c; i = D[i]) {

for(int j = R[i]; j != i; j = R[j]) remove(col[j]);

dance(d + 1);

for(int j = L[i]; j != i; j = L[j]) resume(col[j]);

}

resume(c);

}

}DLX;

int main() {

int cases;

int n;

int m;

int p;

int x1, x2, y1, y2;

std::cin >> cases;

if(cases > 10 || cases < 1) return -1;

for(int i = 0; i < cases; i++) {

std::cin >> n >> m >> p;

if(n < 1) return -2;

if(m > 30) return -3;

if(p > 100 || p < 1) return -4;

DLX.init(p,n \* m);

for(int k = 1; k <= p; k++) {

std::cin >> x1 >> y1 >> x2 >> y2;

for(int q = x1; q < x2; q++) {

for(int j = y1 + 1; j <= y2; j++) DLX.link(k, q \* m + j);

}

}

DLX.dance(0);

std::cout << DLX.getAnswer() << std::endl;

}

return 0;

}