Гунин Игорь Павлович

Балл: 112

Задача №1

Условие

На складе хранится K болтов M12x60 и L шпилек M12x40. Для изготовления одного изделия требуется P крепёжных элементов, из них не менее трёх болтов.

Определите, можно ли использовать весь имеющийся крепёж при изготовлении изделий данного вида.
Входные данные: 3 целых положительных числа - K, L, P, каждое меньше 1000, записанные через пробел.

Выходные данные: возможное количество изготовленных изделий либо 0, если использовать весь крепёж не получится.

Исходные данные	Результат
10 110 40	3
15 150 20	0

Исходный код

```
#include <bits/stdc++.h>

typedef long long ll;
typedef long double ld;
typedef unsigned long long ull;
typedef __int128 lll;
typedef unsigned __int128 ulll;

int main(void) {
    std::ios_base::sync_with_stdio(0);
    std::cin.tie(nullptr);
    std::cout.tie(nullptr);
    size_t k, l, p;
    std::cin >> k >> l >> p;
    if((k + l) % p) {
        std::cout << 0;
        return 0;
    }
    size_t a = (k + l) / p;
    if(k / a < 3) {
        std::cout << 0;
        return 0;
    }
    std::cout << a;
    return 0;
}
```

Задача №2

Условие

Один из производителей электроники, фирма "Хардстронг", разработала прототип нового вычислительного устройства. Оно позволяет обрабатывать 16-разрядные целые положительные числа и способно выполнять над ними 3 операции: побитовый сдвиг влево на k разрядов (k <= 15), вычитание 1024 и вычитание 1.

В случае побитового сдвига старшие разряды числа, выходящие за пределы разрядной сетки, теряются. Младшие разряды заполняются нулями. Например, сдвиг числа 3 на 2 разряда даст новое значение 12, а сдвиг числа 8193 на 7 даст 128.

В случае получения в результате какой-либо операции отрицательного числа либо нуля выполнение программы завершится исключением (ошибкой).

Выполнение каждой из трёх операций занимает определённое время, не зависящее от значений операндов: сдвиг - 4 такта, вычитание 1024 - 2 такта, вычитание 1 - один такт.

Для заданных целых положительных чисел A и B, каждое меньше 65536, требуется определить минимальное количество тактов для получения числа B из числа A. В случае, если с помощью описанного вычислителя число B из числа A получить невозможно, вывести -1.

Исходные данные	Результат
4 8	4
2049 1024	3

Исходный код

```
#include <bits/stdc++.h>

typedef long long ll;
typedef long double ld;
typedef unsigned long long ull;
typedef __int128 lll;
typedef unsigned __int128 ulll;

int main(void) {
    unsigned short a, b, ans;
    std::cin >> a >> b;
    if(a >= b)
        ans = 0;
    else
        ans = 4;
    for(ll i = 0; i < 15 and a < b; i++, a <<= 1);
    if(a < b) {
        std::cout << -1;
        return 0;
    }
    while(a - b >= 1024) {
        a -= 1024;
        ans += 2;
    }
    while(a > b) {
        a--;
        ans++;
    }
    std::cout << ans;
    return 0;
}
```

Задача №3

Условие

Дано множество отрезков на плоскости. Требуется определить, сколько параллелограммов можно построить из этих отрезков. При этом отрезки можно перемещать, но нельзя менять их размер и угол наклона к осям. Один отрезок можно включать только в один параллелограмм.

Входные данные: в первой строке натуральное N < 100 - количество отрезков. Далее в каждой строке по 4 целых числа, записанных через пробел: x1, y1, x2, y2 - координаты начала и конца отрезка.

Выходные данные: натуральное число - количество параллелогораммов.

Исходные данные	Результат
5 0 0 5 0 0 1 5 1 0 0 1 1 1 1 2 2 1 2 5 2	1
4 0 0 0 2 0 0 1 2 0 0 2 2 0 0 2 0	0

Исходный код

```
#include <bits/stdc++.h>

typedef long long ll;
typedef long double ld;
typedef unsigned long long ull;
typedef __int128 lll;
typedef unsigned __int128 ulll;

union line {
    ld l, tga;
    line(ld _l, ld _tga) {
        l = _l;
        tga = _tga;
    }
    line() {
        l = 0;
        tga = 0;
    }
};

inline bool equals(ld a, ld b, ld eps) {
    return std::abs(a - b) < eps;
}

static const ld eps = 1e-9l;

inline bool operator==(const line &a, const line &b) {
    return equals(a.l, b.l, eps) and equals(a.tga, b.tga, eps);
}

int main(void) {
    std::ios_base::sync_with_stdio(0);
    std::cin.tie(nullptr);
    std::cout.tie(nullptr);
    size_t n;
    auto line_comp = [](const line &a, const line &b) -> bool {
        return a.l < b.l and a.tga < b.tga;
    };
    std::cin >> n;
    std::set<line, decltype(line_comp)> scope(line_comp);
    std::vector<line> sim_pairs;
    for(ll i = 0; i < n; i++) {
        ld x1, y1, x2, y2;
        std::cin >> x1 >> y1 >> x2 >> y2;
        ld _l = std::hypot(x2 - x1, y2 - y1);
        ld _tga;
        if(x2 - x1 == 0)
            _tga = +INFINITY;
        else
            _tga = (y2 - y1) / (x2 - x1);
        if(scope.count(line(_l, _tga))) {
            scope.erase(line(_l, _tga));
            sim_pairs.push_back(line(_l, _tga));
        }
        else
            scope.insert(line(_l, _tga));
    }
    size_t p = 0;
    for(ll i = 0; i < sim_pairs.size(); i++) {
        for(ll j = i; j < sim_pairs.size(); j++) {
            if(not equals(sim_pairs[j].tga, sim_pairs[i].tga, eps)) {
                sim_pairs.erase(sim_pairs.begin() + i);
                sim_pairs.erase(sim_pairs.begin() + j);
                p++;
            }
        }
    }
    std::cout << p;
    return 0;
}
```

Задача №4

Условие

На вход программе подаётся строка из строчных английских букв длиной < 1000 символов. Требуется выполнить "сжатие" этой строки по следующему принципу: подряд идущие повторяющиеся последовательности символов заменить на цепочки из трёх элементов: число - количество повторов, один экземпляр последовательности и точка.

В результате должна получиться строка наименьшей длины.

Входные данные: строка строчных английских букв.

Выходные данные: "сжатая" строка.

Исходные данные	Результат
aaaabbbbbc	4a.5b.c
abcdbcdbcdef	a3bcd.ef

Исходный код

Задача №5

Условие

Учитель информатики Игорь Петрович включил в программу подготовки 11-го класса изучение языка ассемблера.

Ассемблер - низкоуровневый язык программирования, каждая команда которого прямо соответствует машинной команде (инструкции процессора). Многие команды работают с регистрами - ячейками памяти, расположенными внутри процессора.

На текущий момент ученики Игоря Петровича прошли 3 команды: MOV, ADD, SUB.

Синтаксис этих команд таков:

- MOV <приёмник>, <источник> - копирует содержимое источника в приёмник.
- ADD <приёмник>, <источник> - прибавляет содержимое источника к приёмниу.
- SUB <приёмник>, <источник> - вычитает содержимое источника из приёмника.

Источником для каждой команды может быть числовая константа или название регистра, приёмником - только название регистра.

Доступные регистры - AX, BX, CX, DX. Ёмкость каждого регистра - 16 бит.

Значение регистров на момент начала выполнения программы не определено (то есть заранее неизвестно, может быть любым).

Для контроля успеваемости учеников Игорю Петровичу нужно оценивать понимание учениками "сложных" моментов в работе процессора. Требуется помочь ему составить программу, которая будет определять возникновение следующих ситуаций в коде на ассемблере: переполнение, переход через 0, использование неопределённых значений.

Переполнением будем называть ситуацию, когда в результате арифметической операции результат превысил 65535 или оказался меньше -32768. При этом старший разряд результата отбрасывается, а младшие остаются в регистре. Например, при прибавлении 65535 + 2 в регистре окажется 1, а при вычитании из -32768 1 в регистре окажется 32767. Отрицательные числа хранятся в виде дополнительного кода.

Переходом через 0 будем считать ситуацию, возникающую при изменении знака приёмника в результате сложения или вычитания.

Вход: фрагмент программы на ассемблере. Первая строка - целое положительное число, означающее количество команд и не превышающее 100. Каждая команда записана отдельной строкой, аргументы разделяются пробелами. Числовые константы лежат в диапазоне от -32768 до 65535.

Выход: 3 числа через пробел - количество случившихся переполнений, переходов через 0, использований неопределённых значений.

Исходные данные	Результат

3
MOV AX, 0
MOV BX, 5
SUB AX, BX
	0 1 0

4
 	
MOV AX, 1000
 	
MOV BX, 65000
 	
ADD AX, BX
 	
SUB CX, 128
	1 0 1

Исходный код

```
#include <bits/stdc++.h>

typedef long long ll;
typedef long double ld;
typedef unsigned long long ull;
typedef __int128 lll;
typedef unsigned __int128 ulll;

int main(void) {
    std::ios_base::sync_with_stdio(0);
    std::cin.tie(nullptr);
    std::cout.tie(nullptr);
    ll n;
    std::cin >> n;
    int a = 0, b = 0, c = 0, d = 0, overflow = 0,  zero_pass = 0, undef_usage = 0;
    bool da = 0, db = 0, dc = 0, dd = 0;
    std::string com, arg0, arg1;
    for(ll i = 0; i < n; i++) {
        std::cin >> com >> arg0 >> arg1;
        int *a0, a1;
        bool *d0, d1;
        arg0.erase(arg0.end() - 1);
        if(arg0 == "AX")
            a0 = &a, d0 = &da;
        else if(arg0 == "BX")
            a0 = &b, d0 = &db;
        else if(arg0 == "CX")
            a0 = &c, d0 = &dc;
        else if(arg0 == "DX")
            a0 = &d, d0 = &dd;
        if(arg1 == "AX")
            a1 = a, d1 = da;
        else if(arg1 == "BX")
            a1 = b, d1 = db;
        else if(arg1 == "CX")
            a1 = c, d1 = dc;
        else if(arg1 == "DX")
            a1 = d, d1 = dd;
        else
            a1 = std::stoi(arg1), d1 = true;
        if(com != "MOV" and not *d0) {
            undef_usage++;
        }
        if(not d1) {
            undef_usage++;
        }
        if(com == "MOV") {
            *a0 = a1;
            *d0 = d1;
        }
        else if(not *d0)
            continue;
        else if(com == "ADD") {
            bool sg = (*a0 >= 0);
            *a0 += a1;
            if(*a0 > 65535) {
                overflow++;
                *a0 -= 65536;
            }
            if(sg != (*a0 >= 0))
                zero_pass++;
            *d0 = d1;
        }
        else if(com == "SUB") {
            bool sg = (*a0 >= 0);
            *a0 -= a1;
            if(*a0 < -32768) {
                overflow++;
                *a0 += 65536;
```

```
            }
            if(sg != (*a0 >= 0))
                zero_pass++;
            *d0 = d1;
        }
    }
    std::cout << overflow << ' ' << zero_pass << ' ' << undef_usage << '\n';
    return 0;
}
```

Задача №6

Условие

Из журнала отправки электронных писем по протоколу SMTP требуется определить адреса, отправка на которые не удалась.

Пример журнала:

mail1@gmail.com
250 bmstu.ru is pleased to meet you
250 cdp@bmstu.ru sender accepted
250 mail1@gmail.com accepting mail from a client address
354 Enter mail, end with "." on a line by itself
!!mail1@gmail.com 250
250 SMTP state reset

----
razin-ii 99@example.com
250 bmstu.ru is pleased to meet you
250 cdp@bmstu.ru sender accepted
555 syntax error in RCPT TO parameters
250 SMTP state reset

----
mail2@yandex.ru
250 bmstu.ru is pleased to meet you
250 cdp@bmstu.ru sender accepted
250 mail2@yandex.ru accepting mail from a client address
354 Enter mail, end with "." on a line by itself
!!mail2@yandex.ru 250
250 SMTP state reset

Каждая запись журнала состоит из нескольких строк. Записи разделяются последовательностью "----------" (10 дефисов). В первой строке записи указан почтовый адрес, на который направляется письмо. Далее идут строки, начинающиеся либо с трёхзначного кода SMTP, либо с группы из 8 пробелов, либо пустые строки. Если один из кодов, связанных с письмом, лежит в диапазоне от 500 до 599, значит, при отправке имела место ошибка. Так, второй адрес в примере содержит пробел, что привело к ошибке 555.

Входные данные: журнал отправки электронных писем. Журнал завершается строкой, состоящей из одного символа - точки.

Выходные данные: список адресов из журнала по порядку, попытка отправки писем на которые завершилась неудачно. Вывести по одному адресу в строке.

Исходный код

```
a = ''
cur_email = ''
alr = False
while a != '.':
    a = input()
    if not len(a):
        continue
    if cur_email == '':
        cur_email = a
        alr = False
    elif a == '----------':
        cur_email = ''
    elif a[0] == '5' and not alr:
        print(cur_email)
        alr = True
```

Задача №7

Условие

Разреженной называется такая матрица, в которой количество ненулевых элементов существенно меньше количества нулевых. Разреженные матрицы можно хранить в форматах специального вида, например, только индексы ненулевых элементов и их значения.

Требуется написать программу, которая для разреженной матрицы большого размера определит суммы элементов в каждом столбце.

Входные данные: в первой строке три натуральных числа N, M и K, записанные через пробел. N - количество строк в разреженной целочисленной матрице, M - количество столбцов, K - количество ненулевых элементов. N и M не превышают 10^6, K меньше N*M и не превышает 10^4. Далее по одному в строке указаны ненулевые элементы этой матрицы. Для представления каждого элемента через пробел записаны индексы его строки и столбца, а также значение. Индексы считаются с единицы, значения элементов по модулю не превышают 1000.

Выходные данные: M чисел по одному в строке - суммы элементов в соответствующих столбцах матрицы.

Исходные данные	Результат
4 3 1 1 1 10	10 0 0

Исходный код

```
#include <bits/stdc++.h>

typedef long long ll;
typedef long double ld;
typedef unsigned long long ull;
typedef __int128 lll;
typedef unsigned __int128 ulll;

int main(void) {
    std::ios_base::sync_with_stdio(0);
    std::cin.tie(nullptr);
    std::cout.tie(nullptr);
    size_t n, m, k;
    std::cin >> n >> m >> k;
    std::vector<ll> sum(m, 0);
    for(ll i = 0; i < k; i++) {
        ll a, b, c;
        std::cin >> a >> b >> c;
        sum[b - 1] += c;
    }
    for(ll i : sum) {
        std::cout << i << '\n';
    }
    return 0;
}
```