Весь исходный код:

https://github.com/Apsapeh/ITMO-2nd_year/tree/main/AlgorithmsAndDataStructures/timus/introduction

1155

Некоторое объяснение:

Разрешены могут быть только дуоны первого и третьего уровня:

Дуоны первого уровня, это соседние клетки

Дуоны третьего уровня это сосед соседей соседей.
A -> B -> C -> D, где A - текущая клетка, D - клетка третьего уровня
Т.к. куб, то у каждой клетки есть только одна соседняя клетка третьего уровня

Решение:

1. Сначала разрешаем 12 возможных дуонов первого уровня, без дубликатов. Т.е. если проверили AB, то BA
проверять не будем

2. Разрешаем 4 дуона третьего уровня

Результаты:

- 0.001 сек
- 376 КБ
- 367 из 2832 (11.02.26 01:30)

Код (1155-stupid.c):

```c
#include <stdio.h>

// Оптимизированный вывод за раз
static char g_out_buffer[4 * 1000 + 1] = {0};
static char* g_buffer_cursor = g_out_buffer;


static inline void print_to_buffer(char cell_1, char cell_2, char op) {
    g_buffer_cursor[0] = cell_1;
    g_buffer_cursor[1] = cell_2;
    g_buffer_cursor[2] = op;
    g_buffer_cursor[3] = '\n';
    g_buffer_cursor += 4;
}


#define FIRST_LEVEL(_a, _b, _ac, _bc)                       \
    do {                                                    \
        if (_a && _b) {                                     \
            --_a;                                           \
            --_b;                                           \
            print_to_buffer(_ac, _bc, '-');                 \
        }                                                   \
    } while (0)

#define THIRD_LEVEL(_a, _b, _ac, _bc, _br1c, _br2c)         \
    do {                                                    \
        if (_a && _b) {                                     \
            --_a;                                           \
            --_b;                                           \
            print_to_buffer(_br1c, _br2c, '+');             \
            print_to_buffer(_ac, _br1c, '-');               \
            print_to_buffer(_bc, _br2c, '-');               \
        }                                                   \
    } while (0)

int main(void) {
    // Инициализация ячеек (камер)

    // clang-format off
    int A, B, C, D, E, F, G, H;
    scanf(
        "%d %d %d %d %d %d %d %d",
        &A, &B, &C, &D, &E, &F, &G, &H
    );
    // clang-format on
    //
```

```
    // В обеих частях, клетки не соседние между друг другом
    if (A + C + F + H != B + D + E + G) {
        printf("IMPOSSIBLE\n");
        return 0;
    }

    while (A | B | C | D | E | F | G | H) {
        FIRST_LEVEL(A, B, 'A', 'B');
        FIRST_LEVEL(A, E, 'A', 'E');
        FIRST_LEVEL(B, C, 'B', 'C');
        FIRST_LEVEL(B, F, 'B', 'F');
        FIRST_LEVEL(C, D, 'C', 'D');
        FIRST_LEVEL(C, G, 'C', 'G');
        FIRST_LEVEL(D, A, 'D', 'A');
        FIRST_LEVEL(D, H, 'D', 'H');
        FIRST_LEVEL(E, F, 'E', 'F');
        FIRST_LEVEL(E, H, 'E', 'H');
        FIRST_LEVEL(F, G, 'F', 'G');
        FIRST_LEVEL(G, H, 'G', 'H');

        THIRD_LEVEL(A, G, 'A', 'G', 'B', 'F');
        THIRD_LEVEL(B, H, 'B', 'H', 'C', 'G');
        THIRD_LEVEL(C, E, 'C', 'E', 'D', 'H');
        THIRD_LEVEL(D, F, 'D', 'F', 'A', 'E');
    }

    printf("%s", g_out_buffer);
    return 0;
}
```

1296

Решение:

Храним текущую сумму и максимальную за всё время сумму.
Изначально они равны 0.

Каждое новое число прибавляем к сумме

- Если число положительное, сумма выросла и стала больше максимального, то обновляем максимальное
- Если число отрицательное и сумма стала меньше нуля, то сбрасываем текущую серию (т.е. зануляем сумму)

Результаты:

- 0.001 сек
- 768 КБ
- **3** из 8313 (11.02.26 1:30)

P.S.:

Сам алгоритм очень простой, его можно записать в чуть ли не в 10 строк кода. Но в таком случае он выполняется супер медленно (0.031 сек). Ясное дело, что проблема в неоптимальном чтении ввода через scanf (неоптимальный парсинг int-ов, много чтений из stdin). По этой причине, код такой, какой есть. Зато с ним удалось достичь 0.001 сек выполнения и попасть на третье место в рейтинге решений.

Код (1296.c):

```c
#include <stdio.h>
#include <stdint.h>

// -30000\n - 7 байт
// 7 * 60001 + 1
static char g_buf[420008];

int main(void) {
    // Читаем весь ввод разом
    const int readed = fread(g_buf, 1, sizeof(g_buf), stdin);
    // Условия останова
    g_buf[readed] = '\n';
    g_buf[readed + 1] = '\0';
    char* cursor = g_buf;

    // Пропускаем первое число, оно нам не нужно
    while (*cursor++ != '\n') {
    }

    register char ch;
    register unsigned char is_minus = 0;
    register int value = 0;

    register int sum = 0;
    register int max_sum = 0;

    // Итерируемся по всему вводу
    while (*cursor) {
        ch = *cursor++;

        switch (ch) {
            // Если это конец строки, значит число закончено и его можно обработать
            case '\n':
                value = is_minus ? -value : value;

                int tmp_sum = sum + value;
                if (tmp_sum > max_sum)
                    max_sum = tmp_sum;

                sum = tmp_sum >= 0 ? tmp_sum : 0;
                value = 0;
                is_minus = 0;
                break;
            // Устанавливаем флаг, если число отрицательное
            case '-':
                is_minus = 1;
                break;
            // Иначе это цифра
            default:
```

```
                value = value * 10 + ch - '0';
        }
    }

    // Граничный случай, проблема с тестом 7
    // Если в конце ввода нет ни новой строки, ни возврата каретки,
    // то принудительно проверяем последнее число
    if (value != 0 || is_minus) {
        value = is_minus ? -value : value;
        int tmp_sum = sum + value;
        if (tmp_sum > max_sum)
            max_sum = tmp_sum;
        sum = tmp_sum >= 0 ? tmp_sum : 0;
    }

    printf("%d\n", max_sum);
}
```