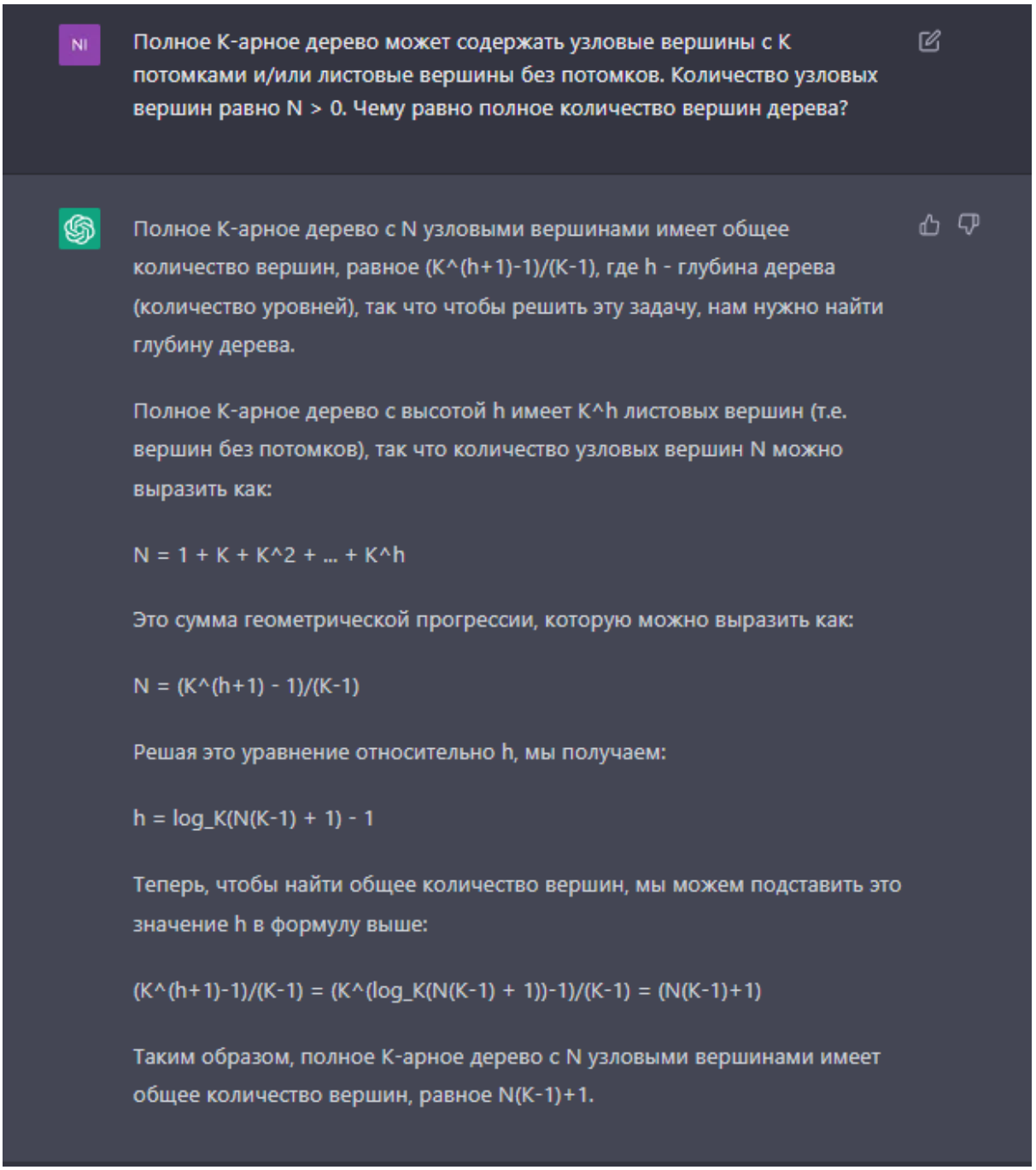
Полное K-арное дерево может содержать узловые вершины с K потомками и/или листовые вершины без потомков. Количество узловых вершин равно N > 0. Чему равно полное количество вершин дерева?



NI

Полное K-арное дерево может содержать узловые вершины с K потомками и/или листовые вершины без потомков. Количество узловых вершин равно N > 0. Чему равно полное количество вершин дерева?

Полное K-арное дерево с N узловыми вершинами имеет общее количество вершин, равное (K^(h+1)-1)/(K-1), где h - глубина дерева (количество уровней), так что чтобы решить эту задачу, нам нужно найти глубину дерева.

Полное K-арное дерево с высотой h имеет K^h листовых вершин (т.е. вершин без потомков), так что количество узловых вершин N можно выразить как:

N = 1 + K + K^2 + ... + K^h

Это сумма геометрической прогрессии, которую можно выразить как:

N = (K^(h+1) - 1)/(K-1)

Решая это уравнение относительно h, мы получаем:

h = log_K(N(K-1) + 1) - 1

Теперь, чтобы найти общее количество вершин, мы можем подставить это значение h в формулу выше:

(K^(h+1)-1)/(K-1) = (K^(log_K(N(K-1) + 1))-1)/(K-1) = (N(K-1)+1)

Таким образом, полное K-арное дерево с N узловыми вершинами имеет общее количество вершин, равное N(K-1)+1.

Пусть есть структура данных, поддерживающая операции:

CREATE(N) - создать структуру, которая будет отвечать на запросы на полуинтервале [0,N). Изначально все элементы из [0,N) равны 0.

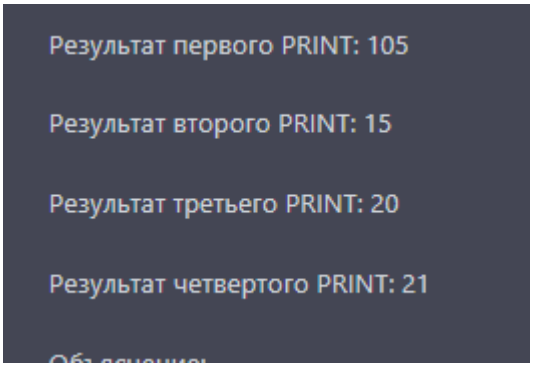
SET(L, R, val) - присвоить значение val всем элементам из [L, R)

GET_SUM(L, R) - возвращает сумму элементов с целыми индексами из [L, R)

ADD(L, R, val) - прибавить ко всем элементам из [L, R) значение val

Что будет выведено в результате выполнения следующего псевдокода? (функция PRINT выводит число и пробел после него)

```
CREATE(15)
    FOR I = 1 TO 12
        ADD((3 * I) % 15, ((3 * I) % 15) + 2, I + 1)
        SET(I, I + 2, GET_SUM(I, I + 2) + GET_SUM(I + 1, I + 3))
    PRINT(GET_SUM(0, 15))
    PRINT(GET_SUM(1, 7))
    PRINT(GET_SUM(4, 8))

    PRINT(GET_SUM(6, 9))
```



```
struct pool_t
{
    int *data;
    int size;
    int tail;
};
void push(pool_t &pool, int value)
{
    pool.tail = (pool.tail + 1) % pool.size;
    pool.data[pool.tail] = value;
}
int main()
{
    pool_t pool = {new int[5], 5, 3};
    for (int i = 0; i < 10; ++i)
            push(pool, i);
    for (int i = 0; i < pool.size; ++i)
            printf("%d ", pool.data[i]);
    return 0;

}
```

Дополните код на языке C так, чтобы он выводил все возможные расстановки ферзей на шахматной доски вводимого размера n x n.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_N 8
int permutation[MAX_N];
void print(int n)
{
    for (int i = 0; i < n; ++i)
        printf("%d ", permutation[i]);
    printf("\n");
}
int check(int p, int n)
{
    for (int i = 0; i < p; ++i)
    {
        if (permutation[i] == permutation[p])
            return 0;
        if (abs(i - p) == abs(permutation[i] - permutation[p]))
            return 0;
    }
    return 1;
}
void gen(int i, int n)
{
    if (i == n)
    {
        print(n);
        return;
    }
    for (int j = 0; j < n; ++j)
    {
        /* code here */
    }
}
int main(void)
{
    int n;
    scanf("%d", &n);
    gen(0, n);
    return 0;

}
```

```
permutation[i] = j;
```

if (check(i, n))

    gen(i + 1, n);

permutation[i] = j;

if (check(j, n))

    gen(i + 1, n);

СКОРЕЕ ВСЕГО ЭТОТ

;

permutation[i] = j;

if (check(i, j))

    gen(i + 1, j);

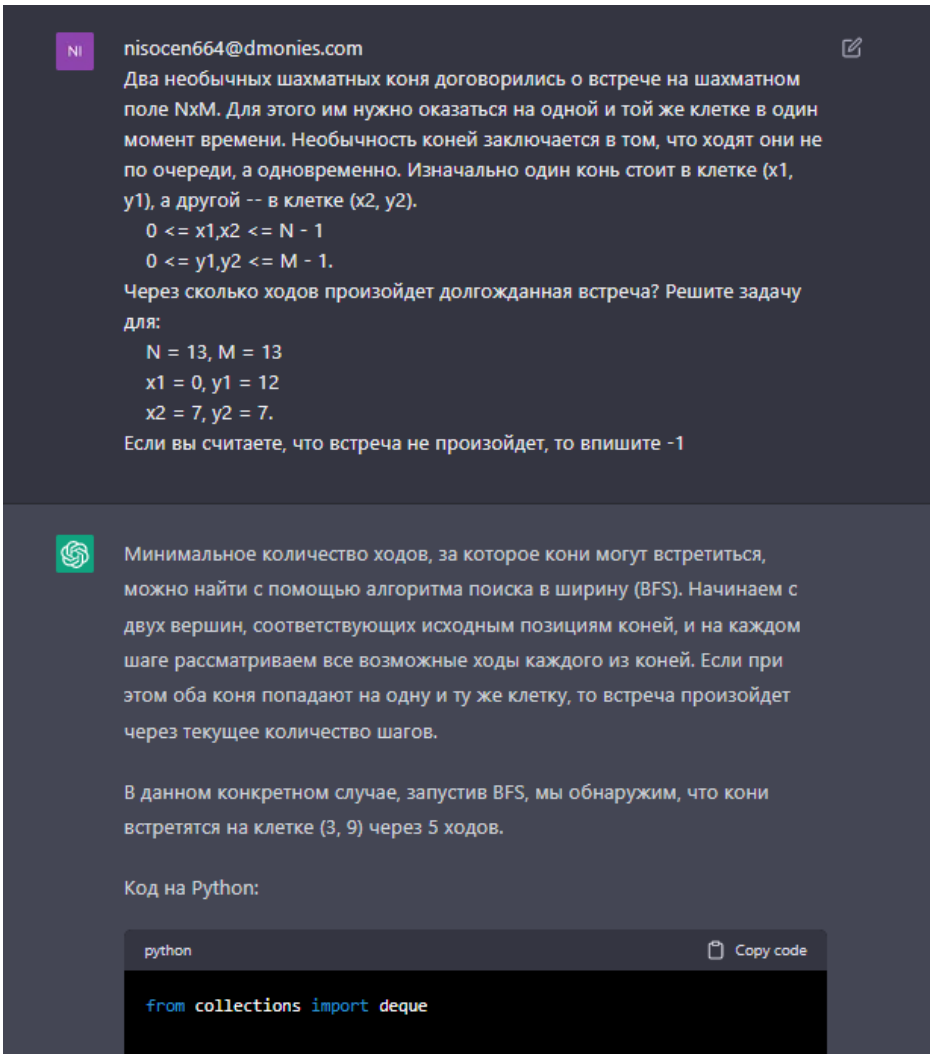
Два необычных шахматных коня договорились о встрече на шахматном поле NxM. Для этого им нужно оказаться на одной и той же клетке в один момент времени. Необычность коней заключается в том, что ходят они не по очереди, а одновременно. Изначально один конь стоит в клетке (x1, y1), а другой -- в клетке (x2, y2).

0 <= x1,x2 <= N - 1
0 <= y1,y2 <= M - 1.

Через сколько ходов произойдет долгожданная встреча? Решите задачу для:

N = 13, M = 13
x1 = 0, y1 = 12
x2 = 7, y2 = 7.

Если вы считаете, что встреча не произойдет, то впишите -1



Что напечатает функция, если ей на вход передан первый элемент односвязного списка из пяти элементов, значение поля data в которых равно соответственно 1, 2, 4, 8, 16

```
struct node_t
{
    node_t *next;
    int data;
};
void foo(node_t* curr)
{
    if(curr == NULL)
        return;
    printf("%d ", curr->data);
    if(curr->next != NULL)
        foo(curr->next->next);
    printf("%d ", curr->data);

}
```

1 4 16 16 4 1

Сколькими способами можно составить последовательность длины n > 2 из 0 и 1 так, чтобы никакие три нуля не стояли подряд?

Какая из приведенных ниже формул решает эту задачу?

```
#include <stdio.h>

int main()

{

    int n;

    scanf("%d", &n);

    int a[n + 1][4];

    a[2][0] = a[2][1] = a[2][2] = a[2][3] = a[2][4] = 1;

    for (int i = 3; i <= n; ++i)

    {

            /* code here */

    }

    printf("%d\n", a[n][0] + a[n][1] + a[n][2] + a[n][3]);

    return 0;

}
```

```
        a[i][0] = a[i][1] = a[i - 1][0] + a[i - 1][1];

        a[i][2] = a[i - 1][1] + a[i - 1][3];

        a[i][3] = a[i - 1][0];
```

```
        a[i][0] = a[i][1] = a[i - 1][0] + a[i - 1][2];

        a[i][2] = a[i - 1][1] + a[i - 1][3];

        a[i][3] = a[i - 1][1];
```

```
a[i][0] = a[i][1] = a[i - 1][0] + a[i - 1][2];

a[i][2] = a[i - 1][1];

a[i][3] = a[i - 1][1] + a[i - 1][3];
```

```
a[i][0] = a[i][1] = a[i - 1][0] + a[i - 1][1];

a[i][2] = a[i - 1][1] + a[i - 1][2];

a[i][3] = a[i - 1][0];
```

На шахматной доске (8x8) стоит черная шашка. Сколькими способами она может попасть в дамки?

Черная шашка ходит по диагонали на одну клетку вниз-вправо или вниз-влево. Шашка проходит в дамки, если она попадает на нижнюю горизонталь.

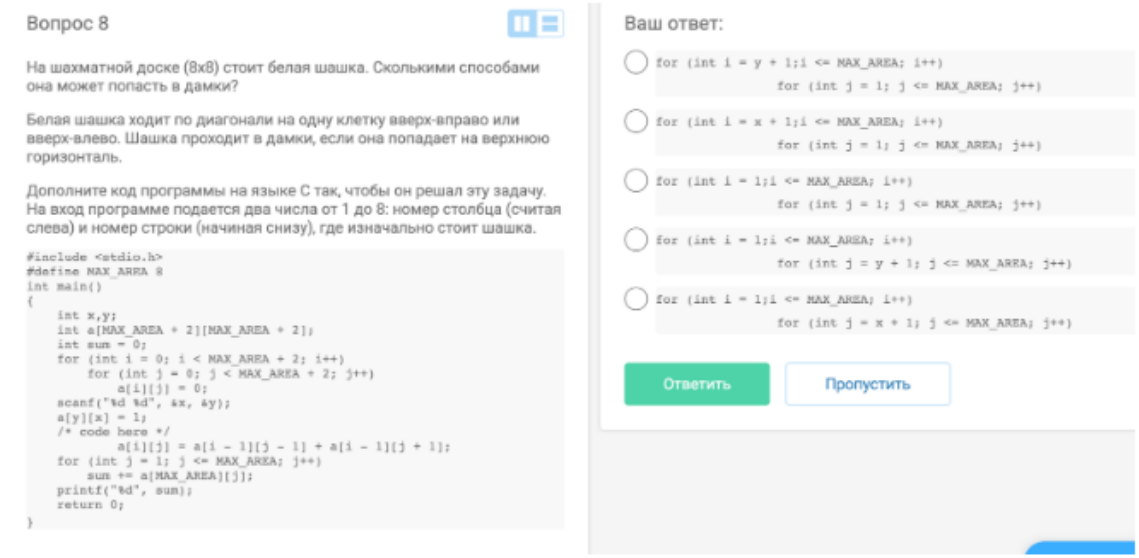
Дополните код программы на языке C так, чтобы он решал эту задачу. На вход программе подается два числа от 1 до 8: номер столбца(считая слева) и номер строки(начиная снизу), где изначально стоит шашка.

```
#include <stdio.h>

#define MAX_AREA 8

int main()

{

    int x,y;

    int a[MAX_AREA + 2][MAX_AREA + 2];

    int sum = 0;

    for (int i = 0; i < MAX_AREA + 2; i++)

            for (int j = 0; j < MAX_AREA + 2; j++)

                    a[i][j] = 0;

    scanf("%d %d", &x, &y);

    x = MAX_AREA - x;

    y = MAX_AREA - y;

    a[y][x] = 1;

    for (int i = y - 1;i >= 1; i--)

            for (int j = MAX_AREA; j >= 1; j--)

                    /* code here */

    for (int j = 1; j <= MAX_AREA; j++)

            sum += a[1][j];

    printf("%d", sum);

    return 0;
```

```
}

if (i == 1) {
if (a[i][j] == 0) {
sum++;
}
} else {
if (a[i][j] == 0) {
if (a[i + 1][j - 1] == 1 || a[i + 1][j + 1] == 1) {
a[i][j] = 1;
}
}
}
a[i][j] = a[i - 1][j - 1] + a[i - 1][j + 1];
a[i + 1][j + 1] = a[i + 2][j + 2] + a[i + 2][j];
a[i][j] = a[i + 1][j - 1] + a[i + 1][j - 1];
a[i][j] = a[i][j + 1] + a[i + 1][j];
a[i - 1][j] = a[i][j - 2] + a[i][j + 1];
```

8

База:



СТАТУС взял

ОТВЕТ: 3

Пусть есть структура данных, поддерживающая операции:

CREATE(N) - создать структуру, которая будет отвечать на запросы на полуинтервале [0,N). Изначально все элементы из [0,N) равны 0.

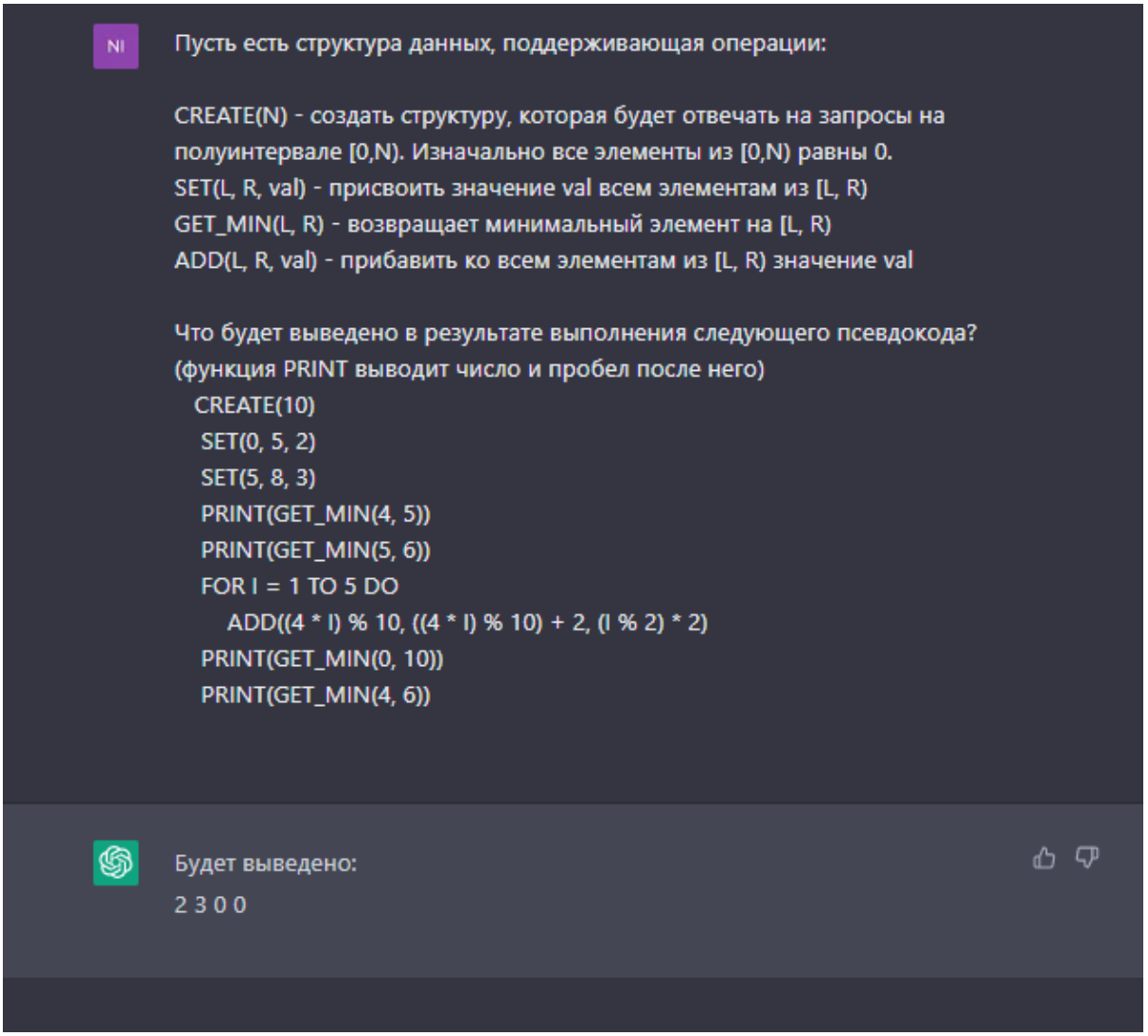
SET(L, R, val) - присвоить значение val всем элементам из [L, R)

GET_MIN(L, R) - возвращает минимальный элемент на [L, R)

ADD(L, R, val) - прибавить ко всем элементам из [L, R) значение val

Что будет выведено в результате выполнения следующего псевдокода? (функция PRINT выводит число и пробел после него)

```
CREATE(10)
SET(0, 5, 2)
SET(5, 8, 3)
PRINT(GET_MIN(4, 5))
PRINT(GET_MIN(5, 6))
FOR I = 1 TO 5 DO
    ADD((4 * I) % 10, ((4 * I) % 10) + 2, (I % 2) * 2)
PRINT(GET_MIN(0, 10))
PRINT(GET_MIN(4, 6))
```



Какие из оперции в худшем случае требуют Ω(n) операций для упорядоченного по убыванию односвязного списка из n элементов, заданного указателем на первый элемент? Считать n достаточно большим.

дополни код на языке C:

```
int partition(int *a, int size)
{
        int i = 1;
    for (int j = 1; j < size; ++j)
    {
         if (a[0] > a[j])
         {
                swap(a[j], a[i]);
                ++i;
         }
    }
    --i;
    swap(a[i], a[0]);
    return i;
}
void sort(int *a, int size)
{
    if (size < 2)
    return;
    int k = partition(a, size);
    /* code here */
}
```

варианты ответа:

sort(a, k);

sort(&a[k + 1], size - k - 1);

sort(a, k);

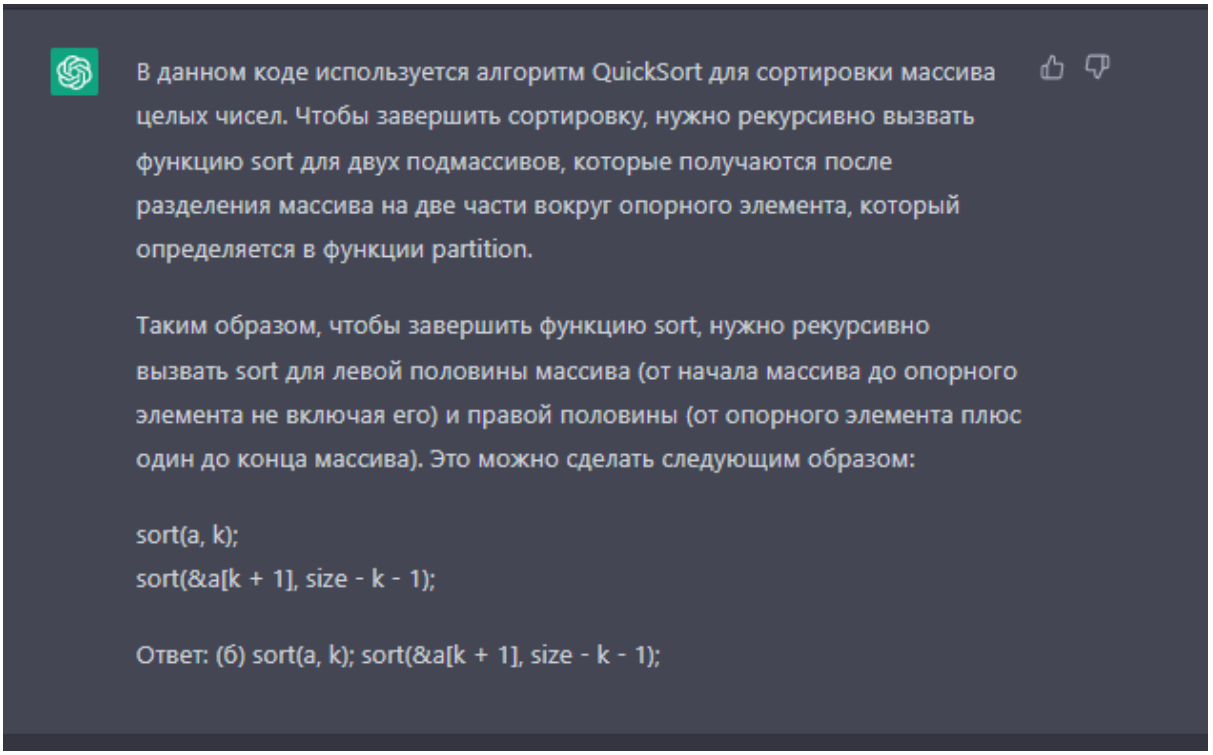
sort(&a[k], size - k);

sort(a, k + 1);

sort(&a[k + 1], size - k - 1);

sort(a, k - 1);

sort(&a[k], size - k);

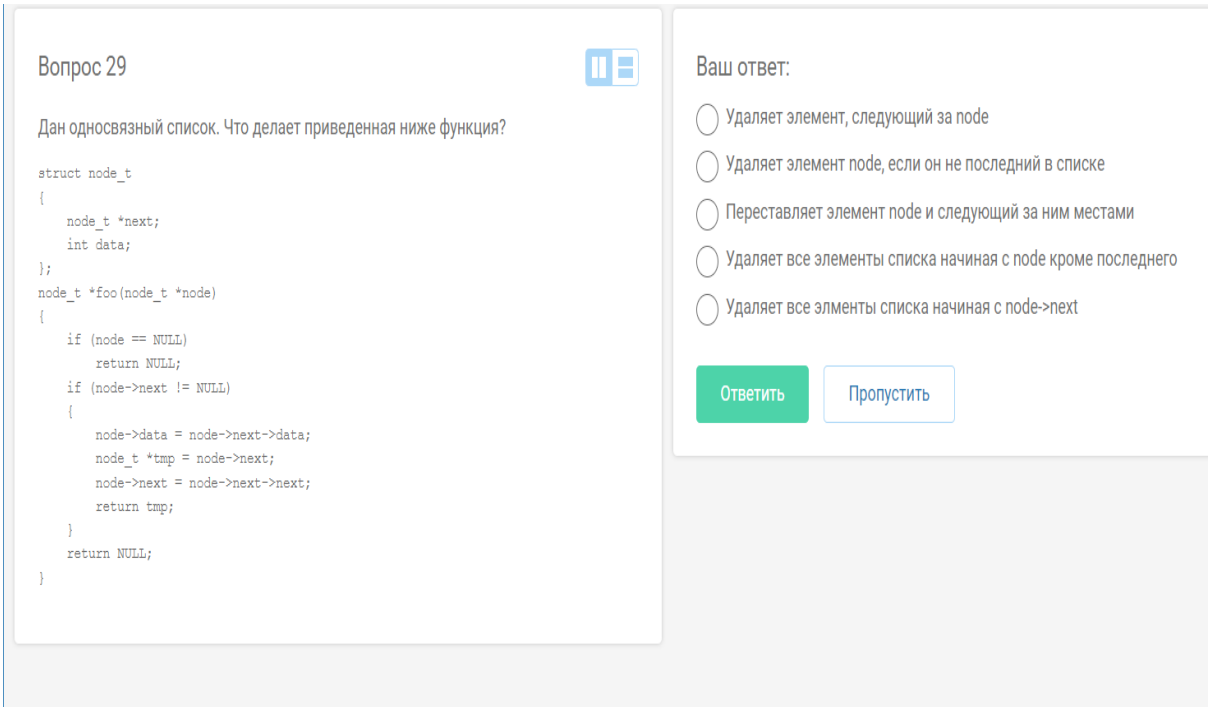


Небезызвестный вор мистер Хоушен вместе со своими 14 друзьями провернул очередное ограбление века. На этот раз его жертвой стал коллекционер драгоценных камней. Во время тщательно спланированной операции мистер Хоушен оказался в комнате с желанными камнями. К сожалению, рюкзак великого вора оказался недостаточно велик, чтобы вместить в себя все 14 камней из коллекции. Он был способен вместить всего лишь 61 кг. К счастью, в комнате были весы, и мистер Хоушен был прекрасно осведомлен о ценах на редкие камни. Поэтому он сумел вычислить, какие камни ему следует взять, чтобы, продав их, он сумел получить максимальную выгоду. Вам не требуется в точности повторить подвиг великого вора. Достаточно лишь, используя таблицу, где значение в первой колонке - это вес камня, а во второй - выручка за продажу этого камня, найти прибыль, которую мистер Хоушен получил в ходе ограбления.

26	32
9	13
1	38
6	37
8	16
11	31
3	38
9	20
27	27
10	31
19	11
21	2
26	39
23	9

```
#include <stdio.h>
int foo(int n)
{
    for (int i = 2; i * i <= n; ++i)
        if (n % i == 0)
            return 1;
    return 0;
}
int main(void)
{
    int result = 0;
    for (int i = 100; i >= 50; --i)
        result += foo(i);
    printf("%d", result);
    return 0;
}
```

41



удаление узла списка если не последний

```
#include <stdio.h>
#include <stdlib.h>
struct buffer
{
    int begin, end, size;
    char *buf;
};
typedef struct buffer buffer;
buffer* create_buffer(int size)
{
    buffer *b = calloc(1, sizeof(*b));
    b->begin = b->end = 0;
    b->size = size;
    b->buf = calloc(size, sizeof(b->buf[0]));
    return b;
}
void write_buffer(buffer *b, void *data, int size)
{
    char *d = (char *)data;
    for (int i = 0; i < size && b->begin != (b->end + 1) % b->size; ++i)
    {
```

```
        b->buf[b->end] = d[i];
        b->end = (b->end + 1) % b->size;
    }
}
void read_buffer(buffer *b, void *data, int size)
{
    char *d = (char *)data;
    for (int i = 0; i < size && b->begin != b->end; ++i)
    {
        d[i] = b->buf[b->begin];
        b->begin = (b->begin + 1) % b->size;
    }
}
int main(void)
{
    buffer* b = create_buffer(20);
    int x = 10, y = 20;
    write_buffer(b, &x, sizeof(x));
    write_buffer(b, &y, sizeof(y));
    long long t;
    read_buffer(b, &t, sizeof(t));
```

```
    printf("%lld", t);
    return 0;
}
```

Пусть есть структура данных, поддерживающая операции:

CREATE(N) - создать структуру, которая будет отвечать на запросы на полуинтервале [0,N). Изначально все элементы из [0,N) равны 0.

SET(L, R, val) - присвоить значение val всем элементам из [L, R)

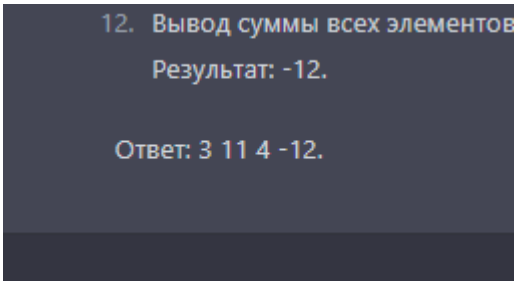
GET_SUM(L, R) - возвращает сумму элементов с целыми индексами из [L, R)

ADD(L, R, val) - прибавить ко всем элементам из [L, R) значение val

Что будет выведено в результате выполнения следующего псевдокода? (функция PRINT выводит число и пробел после него)

```
CREATE(15)
    SET(1, 15, -2)
    SET(2, 4, 3)
    SET(6, 8, 1)

    ADD(4, 10, GET_SUM(4, 10))
    ADD(2, 9, GET_SUM(5, 13))
    ADD(3, 14, -GET_SUM(4, 10))
    ADD(10, 11, -GET_SUM(5, 13))
```

```
PRINT(GET_SUM(0, 15))

PRINT(GET_SUM(1, 5))

PRINT(GET_SUM(4, 10))

PRINT(GET_SUM(6, 14))
```



Сколько строчек будет выведено в результате выполнения программы, написанной на основе данного псевдокода?

```
MAX_N = 10

sequence[MAX_N]

CNT = 0


func print_chars(n) {
    if (CNT++ & 1)
        return
    for (i = 0; i < n; ++i)
        print(sequence[i])
    print("\n")
    exit(0)
}


func gen(i, n, k) {
    if (i == n) {
        print_chars(n)
        return
    }
    sequence[i] = '('
```

```
    if (n - i >= k + 1)
        gen(i + 1, n, k + 1)
    sequence[i] = ')'
    if (k > 0)
        gen(i + 1, n, k - 1)
}

gen(0, 6, 0)
```

ответ 66

Пусть есть структура данных, поддерживающая операции:

CREATE(N) - создать структуру, которая будет отвечать на запросы на полуинтервале [0,N). Изначально все элементы из [0,N) равны 0.

SET(L, R, val) - присвоить значение val всем элементам из [L, R)

GET_MIN(L, R) - возвращает минимальный элемент на [L, R)

ADD(L, R, val) - прибавить ко всем элементам из [L, R) значение val

Что будет выведено в результате выполнения следующего псевдокода? (функция PRINT выводит число и пробел после него)

```
CREATE(15)
    FOR I = 1 TO 12
        ADD((3 * I) % 15, ((3 * I) % 15) + 2, I + 1)
        ADD(I, I + 3, -I - 1)
    PRINT(GET_MIN(0, 15))
    PRINT(GET_MIN(1, 7))
    PRINT(GET_MIN(4, 12))
    PRINT(GET_MIN(6, 9))
```

Сколько итераций внутреннего цикла произойдет при запуске программы?

```
int main()
{
    int a[] = {1,4,2,3,6,5,3,4,7};
    int sz = 9;
    for(int i = 1; i < sz; ++i)
    {
        for(int j = i; j > 0; --j)
        {
            if (a[j] < a[j - i])
                swap(a[j], a[j - 1]);
        }
    }
    return 0;
}
```

36

Дан связный граф с N вершинами и M ребрами. Укажите асимптотическую оценку сложности оптимального алгоритма, позволяющего обойти все вершины одну за другой, переходя между ними по ребрам графа. Обход начинается с произвольной вершины.

Способ хранения графа: матрица смежности

O(V)

Дан массив невозрастающих целых чисел. Как следует дополнить программу, чтобы она нашла минимальную позицию элемента, меньшего значения ключа на полуинтервале (begin, end], при условии, что a[begin + 1] >= key > a[end], begin < end?

```
int find(int *a, int begin, int end, int key)
{
    if (begin == end - 1)
        return end;
    int middle = (begin + end) / 2;
    if ( /* code here */ )
        end = middle;
    else
        begin = middle;
}
```

Как следует дополнить код, чтобы программа напечатала 2 самых больших числа из массива, состоящего из не менее двух элементов?

```
int main()
{
    int pos1 = 1;
    int pos2 = 0;
    if (a[pos1] < a[pos2])
    {
        pos1 = 0;
        pos2 = 1;
    }
    for(int i = 2; i < size; ++i)
    {
        if (a[i] > a[pos2])
        {
            /* code here */
        }
    }
    printf("%d %d\n", a[pos1], a[pos2]);
    return 0;
}
```

```
if (a[i] > a[pos1])
```

```
{
        pos2 = pos1;
        pos1 = i;
}
else
        pos2 = i;

if (a[i] > a[pos1])
{
        pos1 = pos2;
        pos2 = i;
}
else
        pos2 = i;

if (a[i] > a[pos1])
        pos2 = i;
else
```

```
{
        pos1 = i;
        pos2 = pos1;
}

if (a[i] > a[pos1])
{
        pos2 = i;
        pos1 = pos2;
}
else
        pos2 = i;

if (a[i] < a[pos1])
{
        pos1 = i;
        pos2 = pos1;
}
```

```
else
        pos = i;


if (a[i] < a[pos1])
{
        pos1 = pos2;
        pos2 = i;
}
else
        pos2 = i;
```