Практический тур №2

Задача 1 «Числовая лестница»

(уровень сложности: Муниципальный, 9-11 кл, легкая)

Условие задачи:

Натуральные числа расположены в виде числовой лестницы, начиная с 1: на первой строке одно число, на второй – два числа, на следующей – три и так далее.

1					
2	3				
4	5	6			
7	8	9	10		
11	12	13	14	15	
...	...	...	...	...	...

Затем в каждой строке удаляются все числа так, чтобы в ней остались только первые K чисел. Если в строке изначально менее K чисел, то эта строка не изменяется.

Для заданных чисел A, B и K необходимо вывести все строки с номерами от A до B включительно, которые будут получены в результате такого удаления.

Анализ решения

Чтобы вывести требуемый отрезок чисел, нужно вычислить его начало (далее числа идут по порядку). Начало вычисляется по формуле арифметической прогрессии (i-1)*i/2+1 (+1, т.к. нумерация не с нуля).

Полезно также обозначить отдельной переменной количество выводимых в каждой строке чисел. Использование типа данных **unsigned long long** можно рекомендовать в случаях, когда есть вероятность, что результаты будут выражаться большими числами (не умещающимися в стандартный целочисленный тип :)

Текст программы

```cpp
#include <iostream>

int main()
{
    unsigned long long a, b, k;
    std::cin >> a >> b >> k;

    for (unsigned long long i = a; i <= b; ++i)
    {
        // Сколько выводим в одной строке
        unsigned long long u = (i < k) ? i : k;
        // Начало выводимой строки
        unsigned long long res = 1 + i * (i - 1) / 2;
        // Сам вывод
        while (u--)
            std::cout << res++ << ' ';
        std::cout << "\n";
    }
}
```

Задача 2 «Точное время»

(уровень сложности: Муниципальный, 9-11 кл, простая)

Условие задачи:

Чтобы компьютер мог установить у себя точное время, он может использовать специальные сервера, рассылающие значения точного времени. Но при этом просто запросить значение времени у сервера недостаточно, так как данные передаются через сеть с определенной задержкой, и пока значение текущего времени дойдет от сервера до компьютера, оно потеряет свою актуальность.
Поэтому разработан специальный протокол, определяющий взаимодействие клиента (запрашивающего значение времени компьютера) и севера (рассылающего значение времени компьютера), содержащий следующие шаги:

1. Клиентский компьютер отсылает серверу свой запрос и сохраняет (по времени клиента) момент отправления A этого запроса;
2. В момент получения запроса клиента севером его точные часы показывают B. Это значение сервер и отправляет клиенту;
3. Ответ сервера приходит клиенту в момент C по клиентскому времени, это значение клиентом также сохраняется. Теперь он в состоянии установить значение точного времени, располагая известными значениями A, B и C.

Предполагается, что значения задержки при передаче данных в направлениях клиент-сервер и сервер-клиент совпадают.

Требуется реализовать алгоритм, который с точностью до одной секунды находит точное значение времени для установки на клиентском компьютере по известным значениям A, B и C. При получении дробного результата необходимо округлить его до целого числа секунд по обычным правилам арифметики (если дробная часть числа меньше 0.5, то в меньшую сторону, иначе – в большую)

Нужно принять во внимание, что пока клиент ожидает ответа, по его клиентскому времени могут начаться новые сутки. При этом известно, что между моментом отправки запроса и получением ответа от сервера проходит менее 24 часов

Анализ решения

Идея решения довольно проста: (C-A)/2+B.
Т.е. мы находим середину временного отрезка по времени клиента и прибавляем точное время сервера в момент этой середины. Есть несколько нюансов:

- Для удобства всё сразу переводим в секунды, тогда можно пользоваться обычной арифметикой.
- Переход через сутки. Тогда C-A будет отрицательным. Чтобы это скорректировать, достаточно прибавить 24 часа (24*3600 сек).
- Деление на 2 нужно производить корректно (как указано в условии задачи). Нетрудно заметить, что дробная в секундах может быть либо 0.5, либо 0, т.е. округлять нужно только в большую сторону.
- В конце результат нужно обратно привести к виду часы:минуты:секунды.

Обратите внимание, что в данной программе удобнее оказывается использовать функции форматного ввода-вывода языка C, чем аналогичные средства C++.

Текст программы

```c
#include <stdio.h>

int main()
{
    // Ввод данных
    int ah, am, as, bh, bm, bs, ch, cm, cs;
    scanf_s("%d:%d:%d", &ah, &am, &as);
    scanf_s("%d:%d:%d", &bh, &bm, &bs);
    scanf_s("%d:%d:%d", &ch, &cm, &cs);

    // Собственно решение
    int A, B, C, D;

    A = ah * 3600 + am * 60 + as;
    B = bh * 3600 + bm * 60 + bs;
    C = ch * 3600 + cm * 60 + cs;
    D = C - A;

    // Переход через 0
    if (D < 0) D += 24 * 3600;
    if (D & 1) ++D;
    D >>= 1; D += B;
    ah = (D / 3600) % 24; D %= 3600;
    am = D / 60; D %= 60;
    as = D;

    // Вывод результата
    printf("%02d:%02d:%02d", ah, am, as);
    return 0;
}
```

Задача 3 «Неквадратные пары»

(уровень сложности: Муниципальный, 9-11 кл, средняя)

Условие задачи:

Целое число x назовем неквадратным, если не существует такого целого числа y > 1, что x делится на y^2 , то есть $x = y^2z$ для какого-либо целого z.
Даны два целых числа L и R. Необходимо определить сколько существует таких пар целых чисел (a, b), что $L \le a < b \le R$, и оба числа a и b, а также их произведение $a \times b$ являются неквадратными.

Анализ решения

Ограничения данной задачи такие (размер диапазона меньше 1000), что можно создать массив подходящих чисел. Дальше достаточно сравнить каждое число с каждым, на предмет неквадратности произведения, что выполняется с помощью поиска НОД их произведения (у неквадратного произведения НОД сомножителей равен 1).

Проверка неквадратности чисел в заданном диапазоне выполнена урезанным алгоритмом факторизации: выполняется перебор делителей. Если после деления число продолжает делиться на этот же делитель, то число это квадратное, прерываем работу. Если число неквадратное, то придётся перебрать все потенциальные делители (к счастью, это быстро).

Текст программы

Функция проверки неквадратности числа может выглядеть следующим образом:

```
//! Число неквадратное?
bool check(int x)
{
    for (int i = 2; i * i <= x && x; i++)
    {
        if (!(x % i)) x /= i;
        if (!(x % i))
            return false;
    }
    return true;
}
```

Нахождение НОД двух чисел - стандартный алгоритм 😊

```
//! НОД.
int gcd(int x, int y)
{
    int t;
    while (y != 0)
    {
        t = y;
        y = x % y;
        x = t;
    }
    return x;
}
```

И, наконец, главная программа:

```
int main()
{
    // Ввод исходных данных
    int R, L;

    cin >> R >> L;

    // Решение задачи
    int good[1000], cnt = 0, i, j, res;

    for (i = R; i <= L; i++)
        if (check(i))
            good[cnt++] = i;

    for (res = i = 0; i < cnt; i++) for (j = i + 1; j < cnt; j++)
        if (gcd(good[i], good[j]) == 1)
            ++res;

    // Вывод ответа
    cout << res;

    return 0;
}
```

Задача 4 «Не подпоследовательность»

(уровень сложности: Муниципальный, 9-11 кл, сложная)

Условие задачи:

Назовем последовательность $X = (x_1, x_2, \ldots, x_t)$ *подпоследовательностью другой последовательности* Y = (y_1, y_2, …, y_s), если какие-то элементы (возможно ни одного) можно удалить из последовательности Y, чтобы получить последовательность X. Например, последовательность (1, 2, 3, 2) является подпоследовательностью последовательности (1, 1, 2, 2, 1, 3, 2, 1), а последовательность (1, 2, 3, 1, 2) – нет. Заданы две последовательности $A = (a_1, a_2, \ldots, a_m)$ и $B = (b_1, b_2, \ldots, b_n)$, состоящие из целых чисел в диапазоне от 1 до k. Необходимо найти наименьшую по длине последовательность $C = (c_1, c_2, \ldots, c_p)$, которая не являлась бы подпоследовательностью ни A, ни B. Элементы последовательности C также должны являться целыми числами в диапазоне от 1 до k.

Анализ решения

Это последняя (и самая трудная 😊 задача в контесте. Соответственно, приступать к ее решению целесообразно только в случае, когда все предыдущие задачи решены и алгоритмы их решения понятны 😊

Решение начинается с подготовки массивов. Для каждой последовательности важно знать, с какого момента первым встречается тот или иной элемент - это массив first* (индексы первой встречи элемента в последовательности). Второй вспомогательный массив - next*[i,j] - это копия first*, если обработаны последние n-i элементов (если элемента j в окончании массива нет, то сюда записывается некорректный индекс).

Само решение находится проходом слева-направо по этим массивам. bestb/bestbo - позиция во второй последовательности, на которой остановились (позиции в первой - перебираем - это i), очевидно, тут нужно всегда выбирать минимум (два массива за каждый шаг меняются местами, т.о. сохраняется предыдущее состояние, чтобы отсечь неперспективные пути (сравнение на -1)). Процесс кончается, когда будет заполнен bestb/bestbo до конца, т.е. вторая последовательность исчерпана.

Чтобы предъявить искомую последовательность, пришлось сохранять результат выбора на каждом шаге. Для этого ввели два массива: lastc - элементы, fromc - предшествующие позиции (верней, индексы предыдущего элемента).

Для эффективной обработки больших массивов данных (как обычно в последних задачах) нужно оценить целесообразность использования специальных классов-контейнеров, позволяющих быстро выполнять операции, которые на обычных массивах выполняются долго. В данной программе это проиллюстрировано использованием класса **vector** для обработки одномерных и двумерных массивов.

Текст программы

Начало главной функции:

```
// Ввод исходных данных
int m, n, k, a[5001], b[5001], i, j;

cin >> k >> m;
for (i = 0; i < m; i++) { cin >> a[i]; --a[i]; }
cin >> n;
for (i = 0; i < n; i++) { cin >> b[i]; --b[i]; }

// Решение частного случая
if (k == 1)
{
    cout << max(n, m) + 1 << endl;
    for (i = 0; i < max(n, m) + 1; i++)
        cout << "1 ";
    return 0;
}
```

Массивы **first** и **next** создаются следующим образом:

```
vector<vector<int>> nexta(m + 2, vector<int>(k));
for (i = 0; i < k; i++)
    nexta[m + 1][i] = m + 1;

int* firsta = new int[k];
for (i = 0; i < k; i++)
    firsta[i] = m + 1;

for (i = m; i >= 0; i--)
{
    if (i < m) firsta[a[i]] = i + 1;
    for (j = 0; j < k; j++)
        nexta[i][j] = firsta[j];
}

vector<vector<int>> nextb(n + 2, vector<int>(k));
for (i = 0; i < k; i++)
    nextb[n + 1][i] = n + 1;

int* firstb = new int[k];
for (i = 0; i < k; i++)
    firstb[i] = n + 1;

for (i = n; i >= 0; i--)
{
    if (i < n) firstb[b[i]] = i + 1;
    for (j = 0; j < k; j++)
        nextb[i][j] = firstb[j];
}
```

Далее объявляем и инициализируем массивы для формирования результата:

```
int* bestbo = new int[m + 2];
int* bestb = new int[m + 2];
for (i = 0; i < m + 2; i++)
    bestb[i] = -1;
bestb[0] = 0;

vector<vector<int>> lastc(m / k + 1 + n / k + 1, vector<int>(m + 2));
vector<vector<int>> from(m / k + 1 + n / k + 1, vector<int>(m + 2));
int curl = 0;
```

И реализуем основной цикл поиска результата:

```
while (bestb[m + 1] <= n)
{
    int* tmp = bestbo;
    bestbo = bestb;
    bestb = tmp;

    for (i = 0; i < m + 2; i++)
        bestb[i] = -1;

    for (i = 0; i <= m + 1; i++)
    {
        if (bestbo[i] != -1)
        {
            for (int j = 0; j < k; j++)
            {
                int ga = nexta[i][j];
                int gb = nextb[bestbo[i]][j];
                if (bestb[ga] < gb)
                {
                    bestb[ga] = gb;
                    lastc[curl][ga] = j;
                    from[curl][ga] = i;
                }
            }
        }
    }
    curl++;
}
```

Выводим длину получившегося массива и определяем его элементы, затем выводим их:

```
cout << curl << endl;

vector<int> ans;

int curp = m + 1;
while (curl > 0)
{
    ans.push_back(lastc[curl - 1][curp]);
    curp = from[curl - 1][curp];
    curl--;
}

// Вывод результатов

for (i = ans.size() - 1; i >= 0; i--)
    cout << (ans[i] + 1) << " ";
return 0;
```