Разбор задач 2 этапа олимпиады по информатике

Могилёв, 6-8 классы, 2024/2025 уч.г.

A. Голодный бунт (100 баллов)

1. Считываем целое число ***n***.
2. Вычитаем 2 часа (7200 секунд) из ***n*** (по условию $7200 \le n \le 88774$, поэтому результат неотрицательный).
3. Сначала выводим новое время, затем в новой строке напоминание "*Feed the cat!*".

Итоговая сложность: **O(1)**.

A. Голодный бунт (100 баллов)

C++

```cpp
#include <iostream>

using namespace std;

int main()
{
    int n;
    cin >> n;
    int new_time = n - 7200;
    cout << new_time << endl;
    cout << "Feed the cat!";

    return 0;
}
```

Python

```python
n = int(input())
new_time = n – 7200
print(new_time)
print("Feed the cat!")
```

Pascal

```pascal
var n, new_time: longint;
begin
  read(n);
  new_time := n - 7200;
  writeln(new_time);
  writeln('Feed the cat!');
end.
```

B. Маркировка хамелеонов (15 баллов)

Ограничения:

$n = 1$

Нужен 1 отрезок и поскольку рулоны отпускаются целиком, ответ 1 (по условию гарантируется, что длина рулона не менее длины отрезка).

C++

```
#include <iostream>

using namespace std;

int main()
{
    int n, m, d;
    cin >> n >> m >> d;
    cout << 1;
    return 0;
}
```

B. Маркировка хамелеонов (80 и 100 баллов)

1. Поскольку ***d*** задано в метрах, а ***m*** — в сантиметрах, нужно привести длину к сантиметрам: ***d_cm* = *d* * 100** (1 метр = 100 см).
2. Из одного рулона длиной ***d_cm*** можно нарезать ***k* = *d_cm* / *m*** отрезков. Поскольку нельзя склеивать отрезки, это значение округляется вниз (используем целочисленное деление).
3. Общее количество рулонов ***r* = *n* / *k***. Поскольку рулоны отпускаются целиком, результат округляем вверх. Это можно осуществить через целочисленное деление: ***r* = (*n* + *k* – 1) / *k***.

При использовании 32-битного типа данных (**int** в C++, **integer** в Pascal) будет получено 80 баллов. На 100 баллов необходимо использовать 64-битный тип данных (**long long** в C++, **int64** в Pascal).

Итоговая сложность: **O(1)**.

B. Маркировка хамелеонов (100 баллов)

C++

```cpp
#include <iostream>
using namespace std;

int main()
{
    long long n, m, d, d_cm, k, r;

    cin >> n >> m >> d;

    d_cm = d * 100;
    k = d_cm / m;
    r = (n + k - 1) / k;

    cout << r;

    return 0;
}
```

Python

```python
n, m, d = map(int, input().split())
d_cm = d * 100
k = d_cm // m
r = (n + k - 1) // k
print(r)
```

Pascal

```pascal
var n, m, d, d_cm, k, r: int64;
begin
  read(n, m, d);

  d_cm := d * 100;
  k := d_cm div m;
  r := (n + k - 1) div k;

  writeln(r);
end.
```

C. Последовательность (12 баллов)

Ограничения:

$n = 1, m = 1$

1. Считываем целые числа n, m и a_1.
2. Выводим a_1.

C++

```
#include <iostream>
using namespace std;

int main()
{
    int n, m, a;
    cin >> n >> m >> a;
    cout << a;
    return 0;
}
```

C. Последовательность (36 баллов)

Ограничения:

$n = 1, m \le 10^6$

1. Считываем числа n, m и a_1.
2. Для каждого i от 2 до m вычисляем $a_i = i + a_{i-1}$.
3. Выводим a_m.

C++

```
#include <iostream>
using namespace std;

int main()
{
    int n, m, a;
    cin >> n >> m >> a;
    for (int i = 2; i <= m; i++)
        a = i + a;
    cout << a;
    return 0;
}
```

C. Последовательность (44 балла)

Ограничения:

$$n = 1,\ m \le 10^6$$

$$m \le n$$

1. Считываем числа n, m.
2. Если $n = 1$:
   1. Для каждого i от 2 до m вычисляем $a_i = i + a_{i-1}$.
   2. Выводим a_m.
3. Иначе:
   1. Для каждого i от 1 до m считываем a_i.
   2. Выводим a_m.

C++

```
#include <iostream>
using namespace std;
int main()
{
    int n, m, a;
    cin >> n >> m >> a;
    if (n == 1)
    {
        cin >> a;
        for (int i = 2; i <= m; i++)
            a = i + a;
    }
    else
    {
        for (int i = 1; i <= m; i++)
            cin >> a;
    }
    cout << a;
    return 0;
}
```

C. Последовательность (72 балла)

Ограничения:

$m \le 10^6$

1. Считываем числа n, m.
2. Для каждого i от 1 до n считываем a_i.
   1. Если $i = m$, выводим a_m и завершаем программу.
3. Для каждого i от $(n + 1)$ до m вычисляем $a_i = i + a_{i-1}$.
4. Выводим a_m.

C++

```
#include <iostream>
using namespace std;
int main()
{
    long long n, m, a;
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        cin >> a;
        if (i == m)
        {
            cout << a;
            return 0;
        }
    }
    for (int i = n + 1; i <= m; i++)
        a = a + i;
    cout << a;
    return 0;
}
```

C. Последовательность (100 баллов)

Для полного решения нужно уметь быстро вычислять a_m для $m > n$.

Рассмотрим нахождение a_i для i от $(n + 1)$ до m:

$$a_{n+1} = (n + 1) + a_n$$

$$a_{n+2} = (n + 2) + a_{n+1} = (n + 2) + (n + 1) + a_n$$

$$a_{n+3} = (n + 3) + a_{n+2} = (n + 3) + (n + 2) + (n + 1) + a_n$$

…

$$a_{m-1} = (m - 1) + a_{m-2} = (m - 1) + \ldots + (n + 3) + (n + 2) + (n + 1) + a_n$$

$$a_m = m + a_{m-1} = m + (m - 1) + \ldots + (n + 3) + (n + 2) + (n + 1) + a_n$$

C. Последовательность (100 баллов)

$a_m = m + (m - 1) + .. + (n + 3) + (n + 2) + (n + 1) + a_n$

Сумма чисел от **1** до ***n*** равна $\frac{n(n+1)}{2}$, от **1** до ***m*** равна $\frac{m(m+1)}{2}$.

Сумма чисел от **(*n* + 1)** до ***m*** равна $\frac{m(m+1)}{2} - \frac{n(n+1)}{2}$. Тогда:

$a_m = m * (m + 1) / 2 - n * (n + 1) / 2 + a_n$

Итоговая сложность: **O(n)**.

C. Последовательность (100 баллов)

C++

```cpp
#include <iostream>
using namespace std;

int main()
{
    long long n, m, a;
    cin >> n >> m;
    for (int i = 1; i <= n; i++)
    {
        cin >> a;
        if (i == m)
        {
            cout << a;
            return 0;
        }
    }
    cout << a + (1 + m) * m / 2 - (1 + n) * n / 2;
    return 0;
}
```

Python

```python
n, m = map(int, input().split())
a = list(map(int, input().split()))
if m <= n:
    print(a[m - 1])
else:
    print(a[-1] + m * (m + 1) // 2 - n * (n + 1) // 2)
```

C. Последовательность (100 баллов)

Pascal

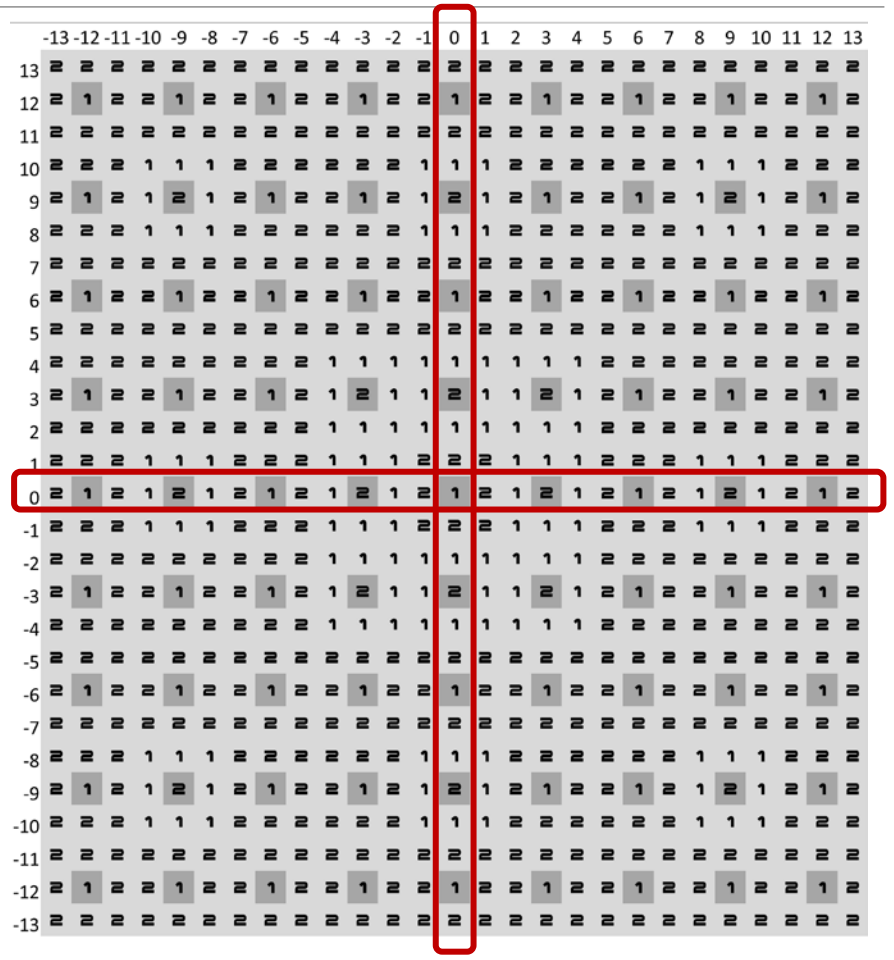
```
var
  i: longint;
  n, m: int64;
  a: array[1..100000] of int64;
begin
  readln(n, m);
  for i := 1 to n do
    read(a[i]);
  if m <= n then
    writeln(a[m])
  else
    writeln(a[n] + m * (m + 1) div 2 - n * (n + 1) div 2);
end.
```

D. Марсианские нанороботы (16 и 20 баллов)

Ограничения:

$y_i = 0$ или $x_i = 0$

Можно заметить, что тип нанороботов **1** и **2** по осям ***Ox*** ($y_i = 0$) и ***Oy*** ($x_i = 0$) чередуется: **1** для чётных ненулевых x_i и y_i, **2** для нечётных.



D. Марсианские нанороботы (16 и 20 баллов)

Нужно учесть, что координаты могут выходить за пределы области, занятой нанороботами.

Каждый день, размер занятого квадрата увеличивается в 3 раза. В первый день **1**, во второй – **3**, …, в день t – 3^{t-1}.

Так как центр занятого квадрата совпадает с началом координат, все заполненные клетки имеют координаты, не превосходящие $3^{t-1}/2$ по абсолютной величине. Остальные клетки пустые.

Для получения 20 баллов, необходимо использовать 64-битный тип данных (**long long** в C++, **int64** в Pascal).

D. Марсианские нанороботы (20 баллов)

C++

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    long long n, t, p = 1, x, y;

    cin >> n >> t;

    for (int i = 1; i < t; i++)
        p *= 3;
    p /= 2;

    for (int i = 0; i < n; i++)
    {
        cin >> x >> y;
        if (abs(x) > p || abs(y) > p) cout << 0;
        else if (y == 0)
        {
            if (x % 2 == 0) cout << 1;
            else cout << 2;
        }
        else if (x == 0)
        {
            if (y % 2 == 0) cout << 1;
            else cout << 2;
        }
    }
    return 0;
}
```

D. Марсианские нанороботы (64 балла)

Ограничения:

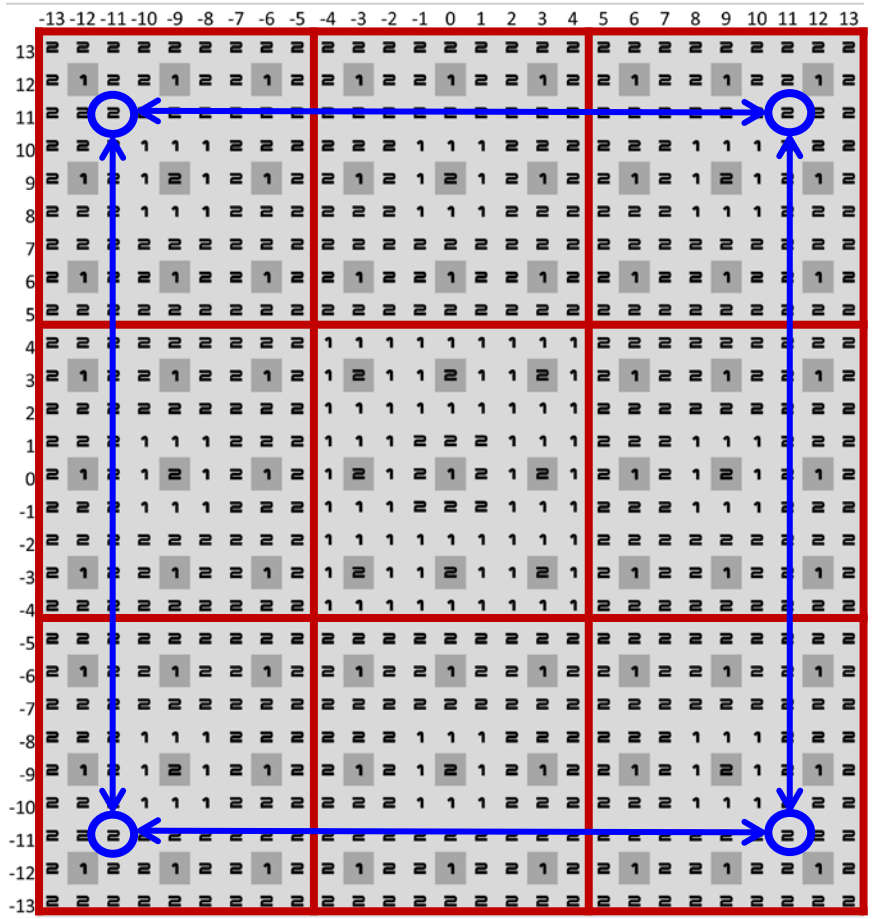
$$-10^3 \le y_i, x_i \le 10^3$$

Можно смоделировать процесс роста колонии нанороботов до **8** дней (размер заполненного квадрата $3^{8-1} = 2187$, **2187/2 = 1093** по осям ***Ox*** и ***Oy***, что больше ограничений).

D. Марсианские нанороботы (100 баллов)

Легко доказать, что занятый нанороботами квадрат симметричен по осям ***Ox*** и ***Oy***, т.е. значения в клетках с координатами $(\pm x_i, \pm y_i)$ совпадают.

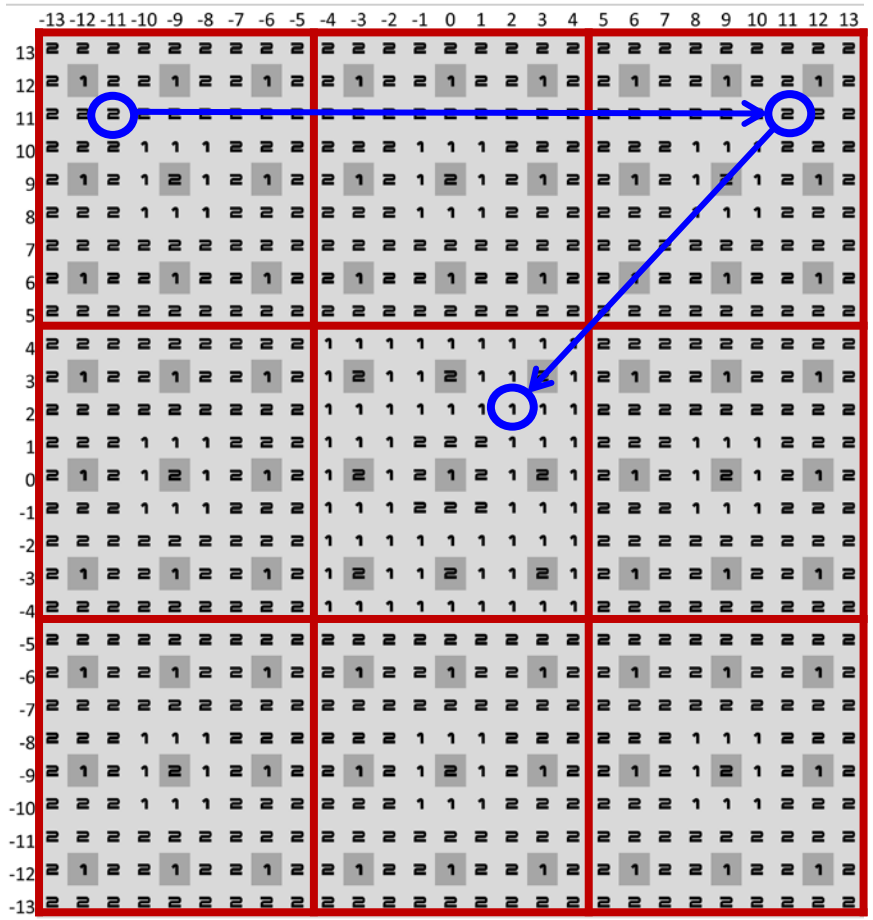
Занятый квадрат можно разделить на 9 квадратов: 8 одинаковых по краям, 1 центральный с инвертированными значениями (т.е. вместо **1** в нём **2** и наоборот).



D. Марсианские нанороботы (100 баллов)

1. Если клетка за пределами занятого квадрата, выводим **0**.
2. Иначе:
    1. Пока не придём в начало координат:
        1. Разобьём квадрат на 9 равных по размеру квадратов.
        2. Если клетка находится в «крайних» квадратах, её можно заменить на симметричную клетку с неотрицательными координатами и тем же значением.
        3. Затем заменить на клетку в центральном квадрате с инвертированным значением (**1** станет **2**, **2** станет **1**).
        4. Переходим к рассмотрению центрального квадрата.
    2. Если количество инверсий чётно, выводим **1**, иначе **2**.

Итоговая сложность: **O(*n* log(max(x_i, y_i)))**.



D. Марсианские нанороботы (100 баллов)

C++

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    long long n, t, x, y, p = 1, a;
    bool f;

    cin >> n >> t;

    for (int i = 1; i < t; i++)
        p *= 3;

    for (int i = 0; i < n; i++)
    {
        cin >> x >> y;
        x = abs(x);
        y = abs(y);
        if (x > p / 2 || y > p / 2) cout << 0;
        else
        {
            f = true;
            a = p;
            while (x || y)
            {
                if (x > a / 2 || y > a / 2)
                {
                    if (x > a / 2) x = abs(x - a);
                    if (y > a / 2) y = abs(y - a);
                    f = !f;
                }
                a /= 3;
            }
            if (f) cout << 1;
            else cout << 2;
        }
    }
    return 0;
}
```

E. Центральная площадь (4 балла)

Ограничения:

$n = 1, m = 1$

Имеется всего одна плитка. Если она белая (**‘W’**) выводим **1 0**, иначе **0 1**.

C++

```
#include <iostream>
using namespace std;

int main()
{
    int n, m;
    char c;

    cin >> n >> m;
    if (n==1 && m==1)
    {
        cin >> c;
        if (c == 'W') cout << 1 << " " << 0;
        else          cout << 0 << " " << 1;
    }
    return 0;
}
```

E. Центральная площадь (12 баллов)

Ограничения:

$n = 1$ или $m = 1$

Имеется всего одна строка или один столбец из плиток. Проверяем наличие хотя бы одной плитки каждого цвета. Если находим, выводим **1**, иначе **0**.

E. Центральная площадь (12 баллов)

C++

```
#include <iostream>

using namespace std;

int main()
{
    int n, m;
    char c;
    bool w = false, r = false;

    cin >> n >> m;
    if (n == 1 || m == 1)
    {
        for (int i = 0; i < n * m; i++)
        {
            cin >> c;
            if (c == 'W') w = true;
            if (c == 'R') r = true;
        }
        if (w) cout << 1 << " ";
        else   cout << 0 << " ";
        if (r) cout << 1;
        else   cout << 0;
    }
    return 0;
}
```

D. Марсианские нанороботы (28 баллов)

Ограничения:

$n, m \leq 100$

Можно перебрать все плитки (i, j) и для каждой рассмотреть все возможные квадраты размерами от **1** до $\min(i, j)$, для которых текущая плитка является нижним правым углом.

Проверим, что плитки на границе квадрата совпадают по цвету с плиткой (i, j).

E. Центральная площадь (28 баллов)

C++

```
#include <iostream>

using namespace std;

int main()
{
    int n, m, w = 0, r = 0;
    string s[2000];

    cin >> n >> m;
    for (int i = 0; i < n; i++)
        cin >> s[i];

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
        {
            for(int t = 0; t <= min(i, j); t++)
            {
                bool f = true;
                for(int z = 0; z <= t; z++)
                {
                    if(s[i][j] != s[i - z][j]) f = false;
                    if(s[i][j] != s[i][j - z]) f = false;
                    if(s[i][j] != s[i - z][j - t]) f = false;
                    if(s[i][j] != s[i - t][j - z]) f = false;
                }
                if (f)
                {
                    if(s[i][j] == 'W') w = max(w, t + 1);
                    else               r = max(r, t + 1);
                }
            }
        }
    cout << w << " " << r;
    return 0;
}
```

E. Центральная площадь (56 баллов)

Ограничения:

$n, m \leq 1000$

Усовершенствуем предыдущий алгоритм.

Будем перебирать квадраты размерами не от **1**, а начиная с размера минимального квадрата для цвета плитки (i, j).

Проверку границ нужно завершать, если встретили плитку не подходящего цвета.

E. Центральная площадь (56 баллов)

C++

```
#include <iostream>
using namespace std;

int main()
{
    int n, m, w = 0, r = 0;
    string s[2000];

    cin >> n >> m;
    for (int i = 0; i < n; i++)
        cin >> s[i];

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
        {
            int q;
            if(s[i][j]=='W') q = w;
            else             q = r;

            for(int t = q; t <= min(i, j); t++)
            {
                bool f = true;
                for(int z = 0; z <= t; z++)
                {
                    if(s[i][j] != s[i - z][j]) {f = false; break;}
                    if(s[i][j] != s[i][j - z]) {f = false; break;}
                    if(s[i][j] != s[i - z][j - t]) {f = false; break;}
                    if(s[i][j] != s[i - t][j - z]) {f = false; break;}
                }
                if (f)
                {
                    if(s[i][j] == 'W') w = max(w, t + 1);
                    else               r = max(r, t + 1);
                }
            }
        }
    cout << w << " " << r;
    return 0;
}
```

E. Центральная площадь (100 баллов)

Для полного решения необходимо уметь определять цвет границы квадрата за O(1). Это можно сделать с помощью префикс сумм или посчитав количество подряд идущих плиток одного цвета по строкам и столбцам.

Итоговая сложность: **O(*nm*)**.

E. Центральная площадь (100 баллов)

C++

```
#include <iostream>
using namespace std;

int hr[2000][2000], hw[2000][2000];
int vr[2000][2000], vw[2000][2000];

int main()
{
    int n, m, w = 0, r = 0;
    string s[2000];

    cin >> n >> m;
    for (int i = 0; i < n; i++)
    {
        cin >> s[i];
        for (int j = 0; j < m; j++)
            if (s[i][j] == 'R')
            {
                hr[i][j] = 1 + (j ? hr[i][j - 1] : 0);
                vr[i][j] = 1 + (i ? vr[i - 1][j] : 0);
                hw[i][j] = 0;
                vw[i][j] = 0;
            }
            else
            {
                hr[i][j] = 0;
                vr[i][j] = 0;
                hw[i][j] = 1 + (j ? hw[i][j - 1] : 0);
                vw[i][j] = 1 + (i ? vw[i - 1][j] : 0);
            }
    }

    for (int i = 0; i < n; i++)
        for (int j = 0; j < m; j++)
            if (s[i][j] == 'W')
            {
                for (int t = w; t <= (min(i, j), min(hw[i][j], vw[i][j])); t++)
                {
                    if (hw[i][j] > t && vw[i][j] > t && hw[i - t][j] > t && vw[i][j - t] > t)
                        w = max(w, t + 1);
                }
            }
            else
            {
                for (int t = r; t <= (min(i, j), min(hr[i][j], vr[i][j])); t++)
                {
                    if (hr[i][j] > t && vr[i][j] > t && hr[i - t][j] > t && vr[i][j - t] > t)
                        r = max(r, t + 1);
                }
            }
    cout << w << " " << r;
    return 0;
}
```