OJ 460

Overlapping Rectangles

Prof. Edson Alves

Faculdade UnB Gama

OJ 460 – Overlapping Rectangles

Problema

When displaying a collection of rectangular windows on a SUN screen, a critical step is determining whether two windows overlap, and, if so, where on the screen the overlapping region lies.

Write a program to perform this function. Your program will accept as input the coordinates of two rectangular windows. If the windows do not overlap, your program should produce a message to that effect. If they do overlap, you should compute the coordinates of the overlapping region (which must itself be a rectangle).

All coordinates are expressed in "pixel numbers", integer values ranging from 0 to 9999. A rectangle will be described by two pairs of (X, Y) coordinates. The first pair gives the coordinates of the lower left-hand corner (X_{LL}, Y_{LL}). The second pair gives the coordinates of the upper right-hand coordinates (X_{UR}, Y_{UR}). You are guaranteed that $X_{LL} < X_{UR}$ and $Y_{LL} < Y_{UR}$.

Entrada e saída

Input

Input will contain several test case. It begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

Each test case consists of two lines. The first contains the integer numbers X_{LL}, Y_{LL}, X_{UR} and Y_{UR} for the first window. The second contains the same numbers for the second window.

Entrada e saída

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

For each set of input if the two windows do not overlap, print the message '`No Overlap`'. If the two windows do overlap, print 4 integer numbers giving the X_{LL}, Y_{LL}, X_{UR} and Y_{UR} for the region of overlap.

Note that two windows that share a common edge but have no other points in common are considered to have '`No Overlap`'.

Exemplo de entradas e saídas

Sample Input

```
1
0 20 100 120
80 0 500 60
```

Sample Output

```
80 20 100 60
```

Solução $O(T)$

- As restrições na entrada simplificam a rotina de interseção entre retângulos

Solução $O(T)$

- As restrições na entrada simplificam a rotina de interseção entre retângulos
- Não é necessário codificar estruturas para representar pontos nem intervalos

Solução $O(T)$

- As restrições na entrada simplificam a rotina de interseção entre retângulos
- Não é necessário codificar estruturas para representar pontos nem intervalos
- Todas as expressões envolvem apenas aritmética inteira

Solução $O(T)$

- As restrições na entrada simplificam a rotina de interseção entre retângulos
- Não é necessário codificar estruturas para representar pontos nem intervalos
- Todas as expressões envolvem apenas aritmética inteira
- Um retângulo com coordenadas negativas será utilizado para indicar o caso onde não há interseção

Solução $O(T)$

- As restrições na entrada simplificam a rotina de interseção entre retângulos
- Não é necessário codificar estruturas para representar pontos nem intervalos
- Todas as expressões envolvem apenas aritmética inteira
- Um retângulo com coordenadas negativas será utilizado para indicar o caso onde não há interseção
- Cada caso de teste pode ser resolvido em $O(1)$, de modo que a solução tem complexidade $O(T)$, onde T é o número de casos de teste

Solução com complexidade $O(T)$

```cpp
struct Rectangle {
    int Px, Py, Qx, Qy;

    Rectangle intersection(const Rectangle& r) const {
        auto xmin = std::max(Px, r.Px);
        auto xmax = std::min(Qx, r.Qx);

        if (xmin >= xmax)
            return { -1, -1, -1, -1 };

        auto ymin = std::max(Py, r.Py);
        auto ymax = std::min(Qy, r.Qy);

        if (ymin >= ymax)
            return { -1, -1, -1, -1 };

        return { xmin, ymin, xmax, ymax };
    }
};
```

Solução com complexidade $O(T)$

```
22
23 int main()
24 {
25     int T;
26     std::cin >> T;
27
28     for (int test = 0; test < T; ++test)
29     {
30         int xl, yl, xu, yu;
31
32         std::cin >> xl >> yl >> xu >> yu;
33         Rectangle r { xl, yl, xu,yu };
34
35         std::cin >> xl >> yl >> xu >> yu;
36         Rectangle s { xl, yl, xu,yu };
37
38         auto ans = r.intersection(s);
```

Solução com complexidade $O(T)$

```
40          if (test)
41              printf("\n");
42
43          if (ans.Px == -1)
44              std::cout << "No Overlap\n";
45          else
46              std::cout << ans.Px << " " << ans.Py << " "
47                  << ans.Qx << " " << ans.Qy << '\n';
48      }
49
50      return 0;
51  }
```