Opracowanie: MUR
Mur

1 Rozwiązanie I

Zauważmy, że jeżeli jakiś fragment muru jest chociaż częściowo zasłonięty przez inny fragment, to on albo jeden z jego sąsiadów jest zasłonięty całkowicie. Dzięki temu spostrzeżeniu możemy ograniczyć się do sprawdzenia, czy wszystkie środki fragmentów muru są widoczne (jeżeli cały fragment jest zasłonięty, to jego środek w szczególności też). W tym celu wystarczy sprawdzić, czy dla każdego środka odcinek łączący kościół z tym środkiem nie przecina żadnego fragmentu muru:

```
function sprawdz_miasto()
begin
    for i := 1 to n do
    begin
        srodek_x = (x[i] + x[i mod n + 1]) / 2.0;
        srodek_y = (y[i] + y[i mod n + 1]) / 2.0;
        for j := 1 to n do
            if (i <> j) and przecina(x[j], y[j], x[j mod n + 1], y[j mod n + 1],
                    srodek_x, srodek_y, kosciol_x, kosciol_y) then
                return "NIE";
    end
    return "TAK";
end

for miasto := 1 to m do
begin
    wczytaj_dane();
    wypisz(sprawdz_miasto());
end
```

gdzie funkcja:

```
przecina(x1, y1, x2, y2, x3, y3, x4, y4);
```

sprawdza, czy odcinki $(x1, y1) - (x2, y2)$ oraz $(x3, y3) - (x4, y4)$ przecinają się.

Pozostaje tylko napisać funkcję `przecina`. Przecięcie dwóch odcinków można sprawdzać na wiele sposobów. Najprostszym jednak i najszerzej wykorzystywanym w informatyce jest sposób oparty na tzw. iloczynie wektorowym.

Iloczyn wektorowy możemy zastosować do sprawdzenia, po której stronie wektora $\overrightarrow{w}$ prowadzącego z początku układu współrzędnych do punktu u leży dany punkt v. Jeżeli punkt v leży po prawej stronie $\overrightarrow{w}$, to wartość $u_x \cdot v_y - u_y \cdot v_x$ jest ujemna, jeśli v leży po lewej stronie $\overrightarrow{w}$, to jest ona dodatnia, zaś jeżeli punkt v leży na prostej będącej przedłużeniem wektora $\overrightarrow{w}$, to wartość ta jest równa zero.

Jak więc sprawdzić, czy dwa odcinki $(x1, y1)-(x2, y2)$ oraz $(x3, y3)-(x4, y4)$ się przecinają? Wystarczy sprawdzić, czy zachodzą naraz dwie własności:

1. Punkty $(x1, y1)$ i $(x2, y2)$ leżą po przeciwnych stronach wektora prowadzącego z punktu $(x3, y3)$ do punktu $(x4, y4)$ ...
2. ...i odwrotnie, czy punkty $(x3, y3)$ oraz $(x4, y4)$ leżą po przeciwnych stronach wektora prowadzącego z punktu $(x1, y1)$ do punktu $(x2, y2)$.

Aby sprawdzić własność (1), przesuńmy początek układu do punktu $(x3, y3)$. Chcemy sprawdzić, czy punkt $(x1 - x3, y1 - y3)$ leży po przeciwnej stronie wektora prowadzącego z $(0, 0)$ do $(x4 - x3, y4 - y3)$ niż punkt $(x2 - x3, y2 - y3)$.

Aby sprawdzić własność (2), należy wykonać tę samą operację, ale zamieniając miejscami punkty $(x1, y1)$ z $(x3, y3)$ oraz $(x2, y2)$ z $(x4, y4)$. Ściślej mówiąc, należy sprawdzić, czy punkt $(x3 - x1, y3 - y1)$ leży po przeciwnej stronie wektora prowadzącego z $(0, 0)$ do $(x2 - x1, y2 - y1)$ niż punkt $(x4 - x1, y4 - y1)$.

Pseudokod funkcji `przecina` może więc wyglądać następująco:

```
function znak(x)
begin
    if x > 0 then return 1;
    if x < 0 then return -1;
    return 0;
end

function iloczyn_wektorowy(ux, uy, vx, vy)
begin
    return ux * vy - vx * uy;
```

```
end

function przecina(x1, y1, x2, y2, x3, y3, x4, y4)
begin
    if znak(iloczyn_wektorowy(x1 - x3, y1 - y3, x4 - x3, y4 - y3))
            = znak(iloczyn_wektorowy(x2 - x3, y2 - y3, x4 - x3, y4 - y3))
        then
            return false;
    if znak(iloczyn_wektorowy(x3 - x1, y3 - y1, x2 - x1, y2 - y1))
            = znak(iloczyn_wektorowy(x4 - x1, y4 - y1, x2 - x1, y2 - y1))
        then
            return false;
    return true;
end
```

Powyższe rozwiązanie sprawdza w sposób bezpośredni, czy środek każdego fragmentu muru, a tym samym także każdy fragment muru, jest widoczny w całości z wieży kościoła. Można to zadanie też rozwiązać podobnie, sprawdzając zamiast środków fragmentów — ich wierzchołki.

Oba te rozwiązania działają jednak zbyt wolno. W każdym z miast n może być rzędu 100 000. Powyższe rozwiązania wykonują dla każdego miasta dwie zagnieżdżone pętle, z których każda przebiega po n wartościach. W sumie więc algorytmy te wykonują około $m \cdot n^2$ (formalnie $O(m \cdot n^2)$) kroków dla wszystkich miast, co w naszym przypadku oznacza liczbę rzędu 10^{11}. Aby jednak czas działania programu mieścił się w limitach czasowych, możemy sobie pozwolić jedynie na liczbę kroków rzędu 10^7. Rozwiązania te otrzymywały więc 70% punktów.

Istnieje jednak szybszy algorytm i, co ciekawe, wcale nie bardziej skomplikowany. Potrzeba w nim jedynie poczynić jedno spostrzeżenie...

2 Rozwiązanie II (wzorcowe)

Spostrzeżenie to polega na odwróceniu sposobu myślenia. Zastanówmy się, jak będzie wyglądało to zadanie z perspektywy nie wszystkowidzącego Jacka, ale kogoś, kto przechadza się w kółko po murze miasta. Otóż wszystkowidzący Jacek widzi cały mur, po którym idzie taka osoba, wtedy i tylko wtedy, gdy ta osoba ma kościół zawsze po tej samej stronie (zawsze po lewej lub zawsze po prawej — w zależności od tego, w którą stronę przechadza się po murze).

Ponieważ umiemy już sprawdzić, po której stronie odcinka leży dany punkt, wystarczy jeżeli przejdziemy w pętli kolejno po wszystkich fragmentach muru i dla każdego z nich sprawdzimy, czy kościół leży po tej samej stronie względem tego fragmentu, co względem następnego:

```
{ Zwraca 1, jeżeli kościół leży po lewej stronie i-tego fragmentu, }
{ -1, jeżeli leży po prawej, zaś 0, jeżeli kościół leży na prostej }
{ zawierającej ten fragment. }
function strona(i)
begin
    x1 = x[i mod n + 1] - x[i];
    y1 = y[i mod n + 1] - y[i];
    xk = kosciol_x - x[i];
    yk = kosciol_y - y[i];
    return znak(iloczyn_wektorowy(x1, y1, xk, yk);
end

function sprawdz_miasto()
begin
    for i := 1 to n do
    begin
        if strona(i) <> strona(i mod n + 1) then
            return "NIE";
    end
    return "TAK";
end

for miasto := 1 to m do
begin
    wczytaj_dane();
    wypisz(sprawdz_miasto());
end
```

To rozwiązanie wykonuje dla każdego miasta tylko jedną pętlę po n liczbach, więc jego złożoność obliczeniowa wynosi $O(m \cdot n)$. Oznacza to liczbę kroków rzędu 10^6, co jest już wynikiem w pełni akceptowalnym.