4.1 節末問題 4.1 的解答

問題 4.1.1

(1) 由於 $\vec{A}+\vec{B}=(2+3,4-9)=(5,-5)$ 答案如下。

- $|\vec{A}|=\sqrt{2^2+4^2}=\sqrt{20}=2\sqrt{5}$（$\sqrt{5}$ 的 2 倍）
- $|\vec{B}|=\sqrt{3^2+(-9)^2}=\sqrt{90}=3\sqrt{10}$（$\sqrt{10}$ 的 3 倍）
- $|\vec{A}+\vec{B}|=\sqrt{5^2+(-5)^2}=\sqrt{50}=5\sqrt{2}$（$\sqrt{5}$ 的 2 倍）

(2) 根據內積公式（→**4.1.4項**）計算如下。

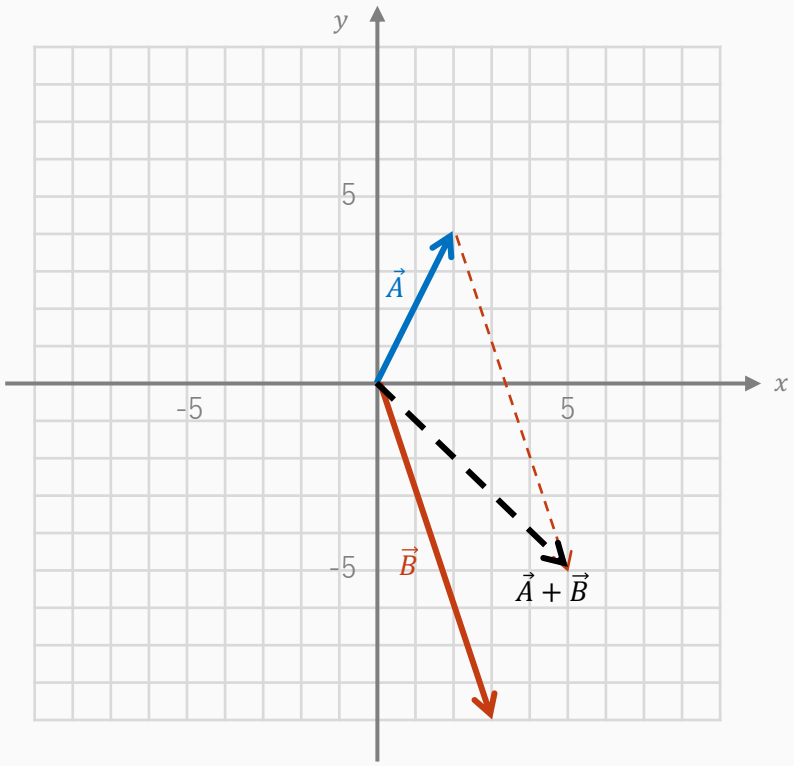
$\vec{A}\cdot\vec{B}=2\times3+4\times(-9)=-30$

(3) 雖然見下圖即可立刻理解，但還是特意利用內積來求解吧。

根據 (2) 的答案，由於內積為負，因此夾角 超過 90 度。

(4) 根據外積公式（→**4.1.5項**）計算如下。

$|A\times B|=|2\times(-9)-3\times4|=30$ 。



問題 4.1.2

點 (x_i, y_i) 與點 (x_j, y_j) 之間的距離可以用下式表示：

$$\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

因此，如下撰寫一個程式將所有點的組合 (i, j) 進行全搜尋，可以得到正解。此外，可以使用 `sqrt` 函式來計算方根。

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

int N;
double x[2009], y[2009];
double Answer = 1000000000.0; // 初始化成非常大的值

int main() {
    // 輸入
    cin >> N;
    for (int i = 1; i <= N; i++) cin >> x[i] >> y[i];

    // 全搜尋
    for (int i = 1; i <= N; i++) {
        for (int j = i + 1; j <= N; j++) {
            // dist 為第 i 個點與第 j 個點之間的距離
            double dist = sqrt((x[i]-x[j]) * (x[i]-x[j]) + (y[i]-y[j]) * (y[i]-y[j]));
            Answer = min(Answer, dist);
        }
    }

    // 輸出答案
    printf("%.12lf\n", Answer);
    return 0;
}
```

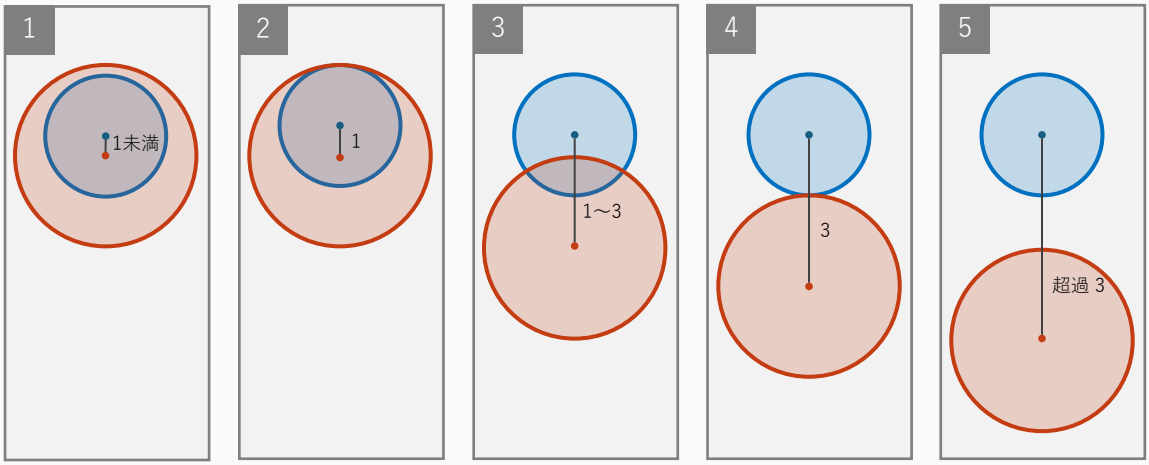
※ Python等原始碼請參閱 chap4-1.md。。

問題 4.1.3

當兩個圓的圓心距離為 d 時，圓的重疊情況如右表所示。（模式的編號請參考書籍的問題敘述）

圓心距離 d	重疊情形
$d < \lvert r_1 - r_2 \rvert$	**模式 [1]**
$d = \lvert r_1 - r_2 \rvert$	**模式 [2]**
$\lvert r_1 - r_2 \rvert < d < r_1 + r_2$	**模式 [3]**
$d = r_1 + r_2$	**模式 [4]**
$r_1 + r_2 < d$	**模式 [5]**

例如，當半徑 2 的圓和半徑 3 的圓逐漸分開時，會如下圖所示。根據前頁的表可知，距離為 1 時在內側相切（**內切**），當距離為 5 時在外側相切（**外切**）。



因此，如下撰寫一個程式來求出圓心距離 d，可以得到正解。計算兩點間距離的方法如節末問題4.1.2所探討。。

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

double X1, Y1, R1;
double X2, Y2, R2;

int main() {
    // 輸入
    cin >> X1 >> Y1 >> R1;
    cin >> X2 >> Y2 >> R2;

    // 求出圓心之間的距離
    double d = sqrt((X1 - X2) * (X1 - X2) + (Y1 - Y2) * (Y1 - Y2));

    // 輸出答案
    if (d < abs(R1 - R2)) cout << "1" << endl;
    else if (d == abs(R1 - R2)) cout << "2" << endl;
    else if (d < R1 + R2) cout << "3" << endl;
    else if (d == R1 + R2) cout << "4" << endl;
    else cout << "5" << endl;
    return 0;
}
```

※ Python等原始碼請參閱 chap4-1.md。

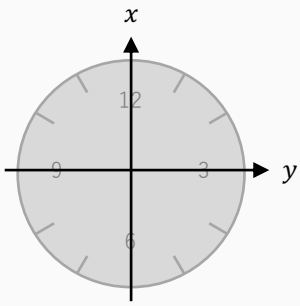
問題 4.1.4

此問題可以使用三角函數（→**專欄 4**）來解決。首先，當12點方向為 0° 時，H 時 M 分的角度如下：

- 時針的角度：$30H + 0.5M°$
- 分針的角度：$6M°$

因此，當時鐘的圓心為座標 $(0, 0)$ 時，各針的座標如下。注意12點方向為 x 軸。

- 時針的前端：$(A\cos(30H + 0.5)°, A\sin(30H + 0.5M)°)$
- 分針的前端：$(B\cos 6H°, B\sin 6H°)$



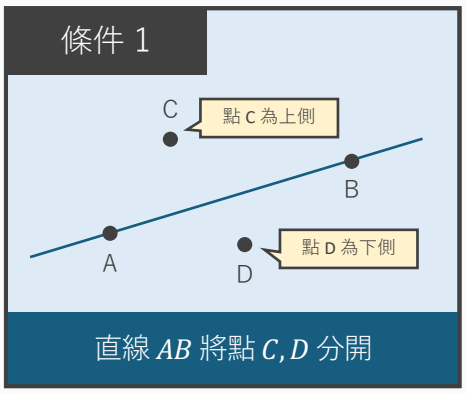
製作計算這兩點間距離的程式即可得到正解。另外，也可以使用餘弦定理（不在本書範圍內）來解決。

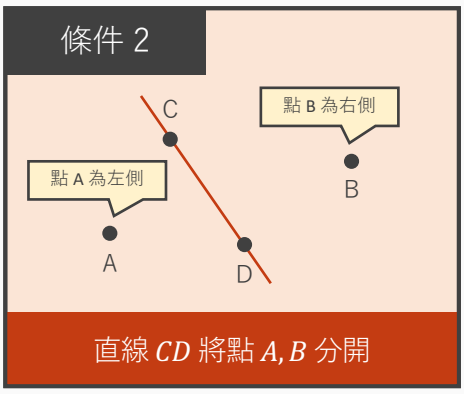
```cpp
#include <iostream>
#include <cmath>
using namespace std;

const double PI = 3.14159265358979;

int main() {
	// 輸入
	double A, B, H, M;
	cin >> A >> B >> H >> M;

	// 求出座標
	double AngleH = 30.0 * H + 0.5 * M;
	double AngleM = 6.0 * M;
	double Hx = A * cos(AngleH * PI / 180.0), Hy = A * sin(AngleH * PI / 180.0);
	double Mx = B * cos(AngleM * PI / 180.0), My = B * sin(AngleM * PI / 180.0);

	// 求出距離 → 輸出
	double d = sqrt((Hx - Mx) * (Hx - Mx) + (Hy - My) * (Hy - My));
	printf("%.12lf\n", d);
	return 0;
}
```

※ Python等原始碼請參閱 chap4-1.md。

問題 4.1.5

令第 1 條線段的端點為 A, B， 第2條線段的端點為 C, D， 兩條線段革相交（具有共同的點）的充要條件基本為滿足以下兩個條件。

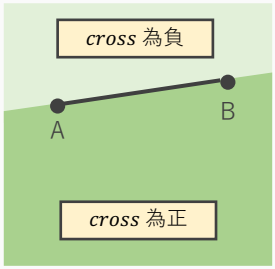




因此， 線段 AB 是否分隔點 C, D， 可以用以下兩個值是否相反來判斷。

- $cross(\overrightarrow{AB}, \overrightarrow{AC})$ 的符號（正負）
- $cross(\overrightarrow{AB}, \overrightarrow{AD})$ 的符號（正負）

cross 函數可以回到 **4.1.5 項**確認。



因此， 如下實作可求解此問題。注意當點 A, B, C, D 排列在一直線上時，$cross(\overrightarrow{AB}, \overrightarrow{AC}) = 0$ 時等特殊狀況（稱為**邊角案例**）， 需要另外區分。

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

long long cross(long long ax, long long ay, long long bx, long long by) {
    // 向量 (ax, ay) 與 (bx, by) 的外積大小
    return ax * by - ay * bx;
}

int main() {
        // 輸入
        long long X1, Y1, X2, Y2, X3, Y3, X4, Y4;
        cin >> X1 >> Y1; // 輸入點 A 的座標
        cin >> X2 >> Y2; // 輸入點 B 的座標
        cin >> X3 >> Y3; // 輸入點 C 的座標
        cin >> X4 >> Y4; // 輸入點 D 的座標
```

```
    // 計算 cross(AB, AC)
    long long ans1 = cross(X2-X1, Y2-Y1, X3-X1, Y3-Y1);
    long long ans2 = cross(X2-X1, Y2-Y1, X4-X1, Y4-Y1);
    long long ans3 = cross(X4-X3, Y4-Y3, X1-X3, Y1-Y3);
    long long ans4 = cross(X4-X3, Y4-Y3, X2-X3, Y2-Y3);

    // 全部排在一直線上時（邊角案例）
    if (ans1 == 0 && ans2 == 0 && ans3 == 0 && ans4 == 0) {
        // 將 A, B, C, D 視為數值（正確來說是 pair 型態）
        // 藉由適當的進行 swap ， 可以假設 A<B, C<D
        // 如此，可以歸結成判斷區間是否重疊的問題（節末問題 2.5.6）
        pair<long long, long long> A = make_pair(X1, Y1);
        pair<long long, long long> B = make_pair(X2, Y2);
        pair<long long, long long> C = make_pair(X3, Y3);
        pair<long long, long long> D = make_pair(X4, Y4);
        if (A > B) swap(A, B);
        if (C > D) swap(C, D);
        if (max(A, C) <= min(B, D)) cout << "Yes" << endl;
        else cout << "No" << endl;
        return 0;
    }

    // 並非如此時
    // IsAB: 線段 AB 是否將點 C, D 分開？
    // IsCD: 線段 CD 是否將點 A, B 分開？
    bool IsAB = false, IsCD = false;
    if (ans1 >= 0 && ans2 <= 0) IsAB = true;
    if (ans1 <= 0 && ans2 >= 0) IsAB = true;
    if (ans3 >= 0 && ans4 <= 0) IsCD = true;
    if (ans3 <= 0 && ans4 >= 0) IsCD = true;

    // 輸出答案
    if (IsAB == true && IsCD == true) {
        cout << "Yes" << endl;
    }
    else {
        cout << "No" << endl;
    }
    return 0;
}
```

※ Python等原始碼請參閱 chap4-1.md。