4.5 節末問題 4.5 的解答

問題 4.5.1

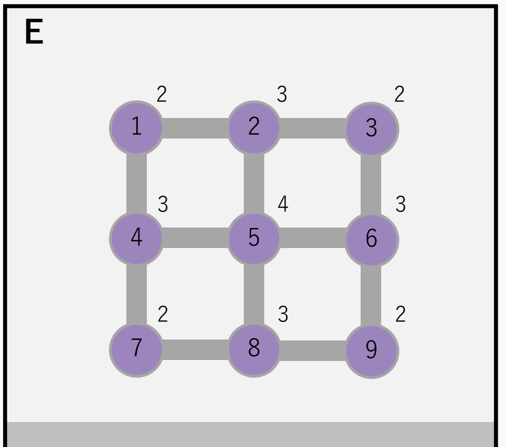
答案如下。不理解的人請確認 4.5.2 項、4.5.4 項。

圖的編號	圖的種類	度最大的頂點
A	無權重無向圖形	頂點 1（度=3）
B	加權無向圖形	頂點 5（度=1）
C	無權重有向圖形	頂點 3（出度=2）
D	加權有向圖形	頂點 2（出度=3）

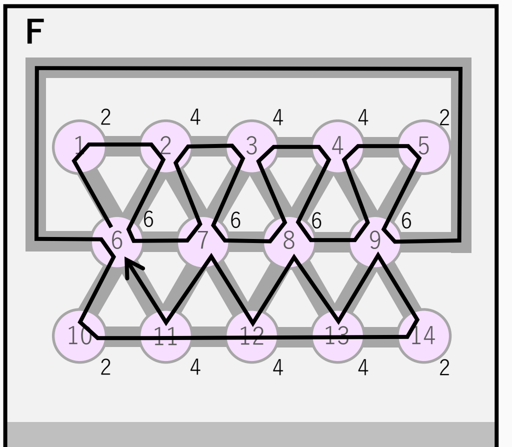
問題 4.5.2

首先，因為圖 E 存在有奇數度的頂點，所以不存在能夠經過一次所有頂點且回到原頂點的路徑。此外，圖F所有頂點的度都是偶數，存在如下圖所示的路徑。

不瞭解的人請確認尤拉圖（→**4.5.2項**）。下圖中頂點所附的數字是該頂點的度。



頂點 2, 4, 6, 8 為奇數度

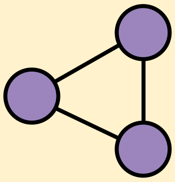


沒有奇數度的頂點

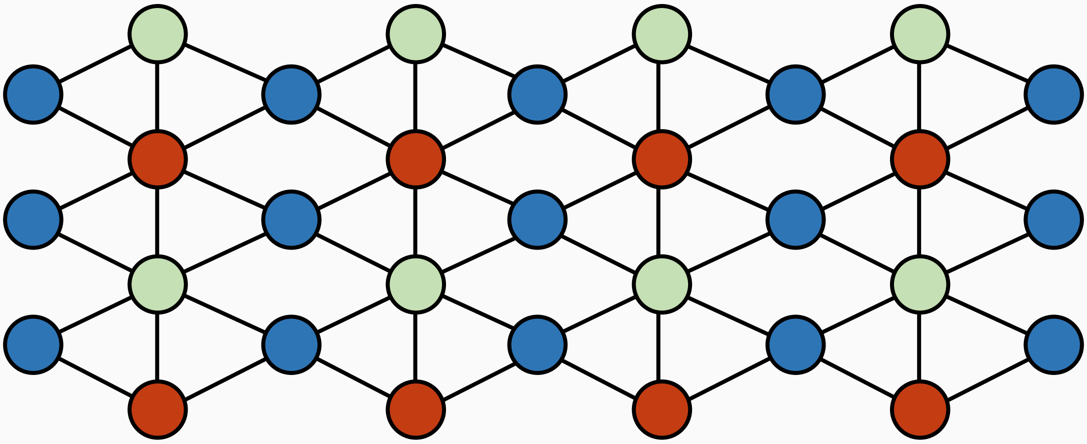
問題 4.5.3

無法用兩種顏色著色的理由如下。

因為圖中包含了如右圖所示的三角形，僅考慮此三角形即可知用兩種顏色著色顯然是不可能的。



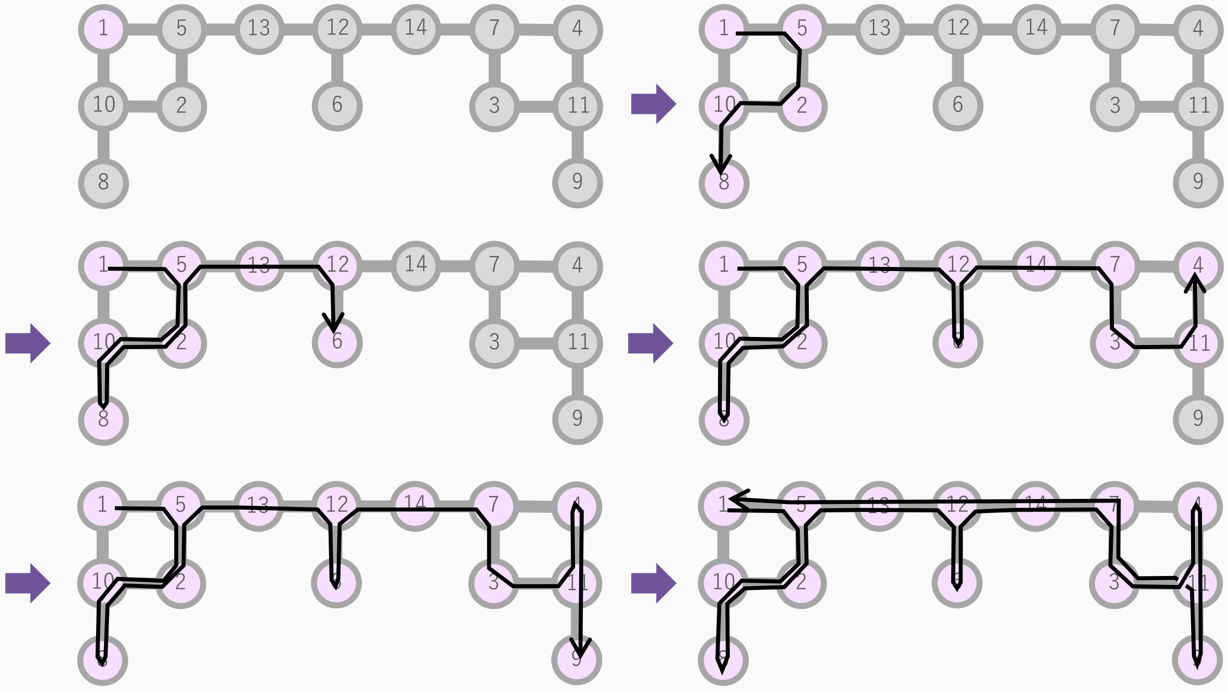
此外，如下圖所示，可以用三種顏色著色。



問題 4.5.4

訪問順序為 1, 5, 2, 10, 8, 13, 12, 6, 14, 7, 3, 11, 4, 9 。

下圖顯示了深度優先搜尋的具體動作。



問題 4.5.5

這是測試對鄰接表形式（→**4.5.5項**）理解的問題。例如進行以下實作即可得到正解。

此外，變數 `cnt` 表示與頂點 i 鄰接的頂點中，編號小於 i 的頂點個數。`G[i].size()` 是列表 `G[i]` 的元素數量。

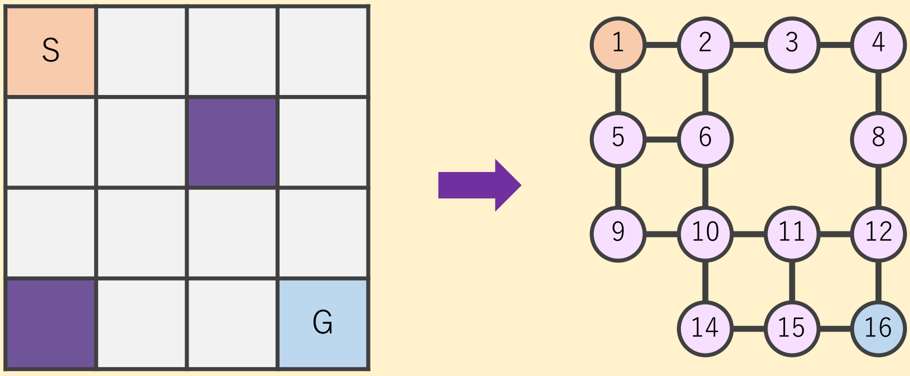
```
#include <iostream>
#include <vector>
using namespace std;

int N, M;
int A[100009], B[100009];
vector<int> G[100009];

int main() {
    // 輸入
    cin >> N >> M;
    for (int i = 1; i <= M; i++) {
        cin >> A[i] >> B[i];
        G[A[i]].push_back(B[i]);
        G[B[i]].push_back(A[i]);
    }

    // 求出答案
    int Answer = 0;
    for (int i = 1; i <= N; i++) {
        int cnt = 0;
        for (int j = 0; j < G[i].size(); j++) {
            // G[i][j] 是與頂點 i 鄰接的頂點中第 j+1 個
            if (G[i][j] < i) cnt += 1;
        }
        // 如果比自己小的鄰接頂點為 1 個的話，將 Answer 加 1
        if (cnt == 1) Answer += 1;
    }

    // 輸出
    cout << Answer << endl;
    return 0;
}
```

※ Python等原始碼請參閱 chap4-5.md。

問題 4.5.6

若設定頂點編號如下，即可歸納為最短路徑問題（→**4.5.7項**）。頂點數為 HW，邊數為 $4HW$ 以下，所以可以在 1 秒內完成執行。。

將從上開始第 i 列，從左開始第 j 個格子的頂點編號設定為 $(i-1)\times W+j$。
$H=4, W=4$ 時的具體例如下圖所示。



因此，實作如下可以得到正解。另外，對各格子標出頂點編號般，將資料用數值來表示以識別的方式稱為雜湊。由於是資格考試等常見的問題，有興趣的人可以在網路等進行調查。

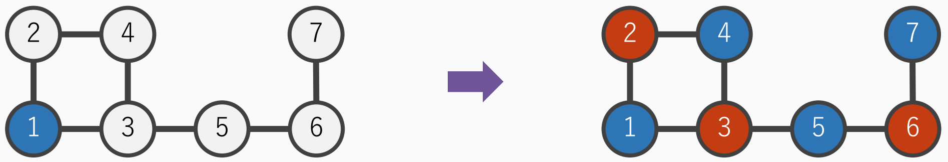
```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

// 輸入
int H, W;
int sx, sy, start; // 起點的座標 (sx, sy) 與頂點編號 sx*H+sy
int gx, gy, goal;  // 終點的座標 (gx, gy) 與頂點編號 gx*W+gy
char c[59][59];

// 圖、最短路徑
int dist[2509];
vector<int> G[2509];

int main() {
    // 輸入
    cin >> H >> W;
    cin >> sx >> sy; start = sx * W + sy;
    cin >> gx >> gy; goal = gx * W + gy;
    for (int i = 1; i <= H; i++) {
        for (int j = 1; j <= W; j++) cin >> c[i][j];
    }
```

```
    //將橫向的邊 [(i, j) - (i, j+1)] 添加到圖中
    for (int i = 1; i <= H; i++) {
        for (int j = 1; j <= W - 1; j++) {
            int idx1 = i * W + j; // 頂點 (i, j) 的頂點編號
            int idx2 = i * W + (j+1); // 頂點 (i, j+1) 的頂點編號
            if (c[i][j] == '.' && c[i][j+1] == '.'){
                G[idx1].push_back(idx2);
                G[idx2].push_back(idx1);
            }
        }
    }

    // 將縱向的邊 [(i, j) - (i+1, j)] 添加到圖中
    for (int i = 1; i <= H - 1; i++) {
        for (int j = 1; j <= W; j++) {
            int idx1 = i * W + j; // 頂點 (i, j) 的頂點編號
            int idx2 = (i+1) * W + j; // 頂點 (i+1, j) 的頂點編號
            if (c[i][j] == '.' && c[i+1][j] == '.'){
                G[idx1].push_back(idx2);
                G[idx2].push_back(idx1);
            }
        }
    }

    // 以下（除了頂點數等）與程式碼 4.5.3 相同
    // 廣度優先搜尋的初始化（dist[i]=-1 時，是未到達的白色頂點）
    for (int i = 1; i <= H * W; i++) dist[i] = -1;
    queue<int> Q; // 定義佇列 Q
    Q.push(start); dist[start] = 0; // 於 Q 添加 1（操作 1）

    // 廣度優先搜尋
    while (!Q.empty()) {
        int pos = Q.front(); // 查看 Q 的開頭（操作 2）
        Q.pop(); // 取出 Q 的開頭（操作 3）
        for (int i = 0; i < (int)G[pos].size(); i++) {
            int nex = G[pos][i];
            if (dist[nex] == -1) {
                dist[nex] = dist[pos] + 1;
                Q.push(nex); // 於 Q 添加 nex（操作 1）
            }
        }
    }

    // 輸出答案
    cout << dist[goal] << endl;
    return 0;
}
```

※ Python等原始碼請參閱 chap4-5.md。

問題 4.5.7

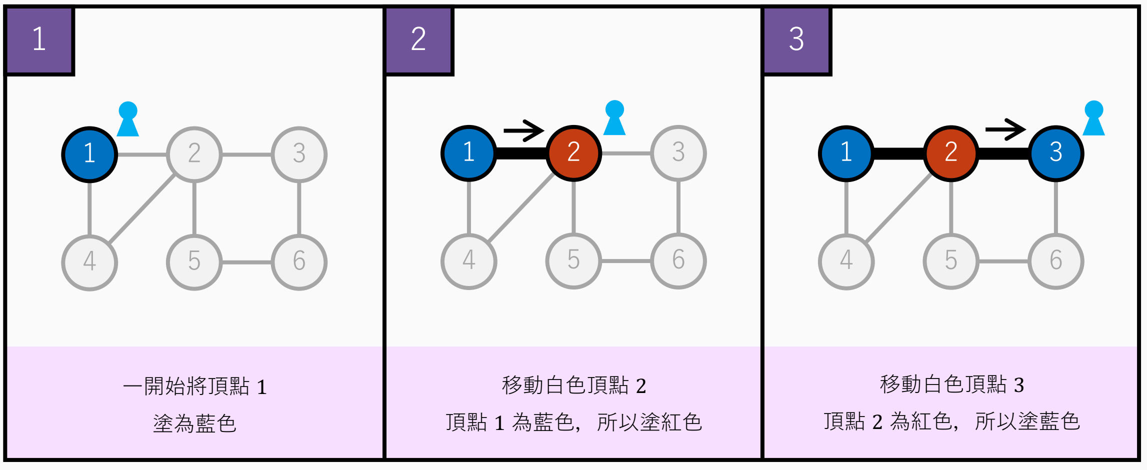
首先，當圖形為連通時，只要決定一個頂點的顏色後，將圖形用藍色和紅色著色的方法最多只有固定一種（如下圖所示）。其原因是必須藍色的旁邊塗紅色、紅色的旁邊塗藍色…如此著色下去。

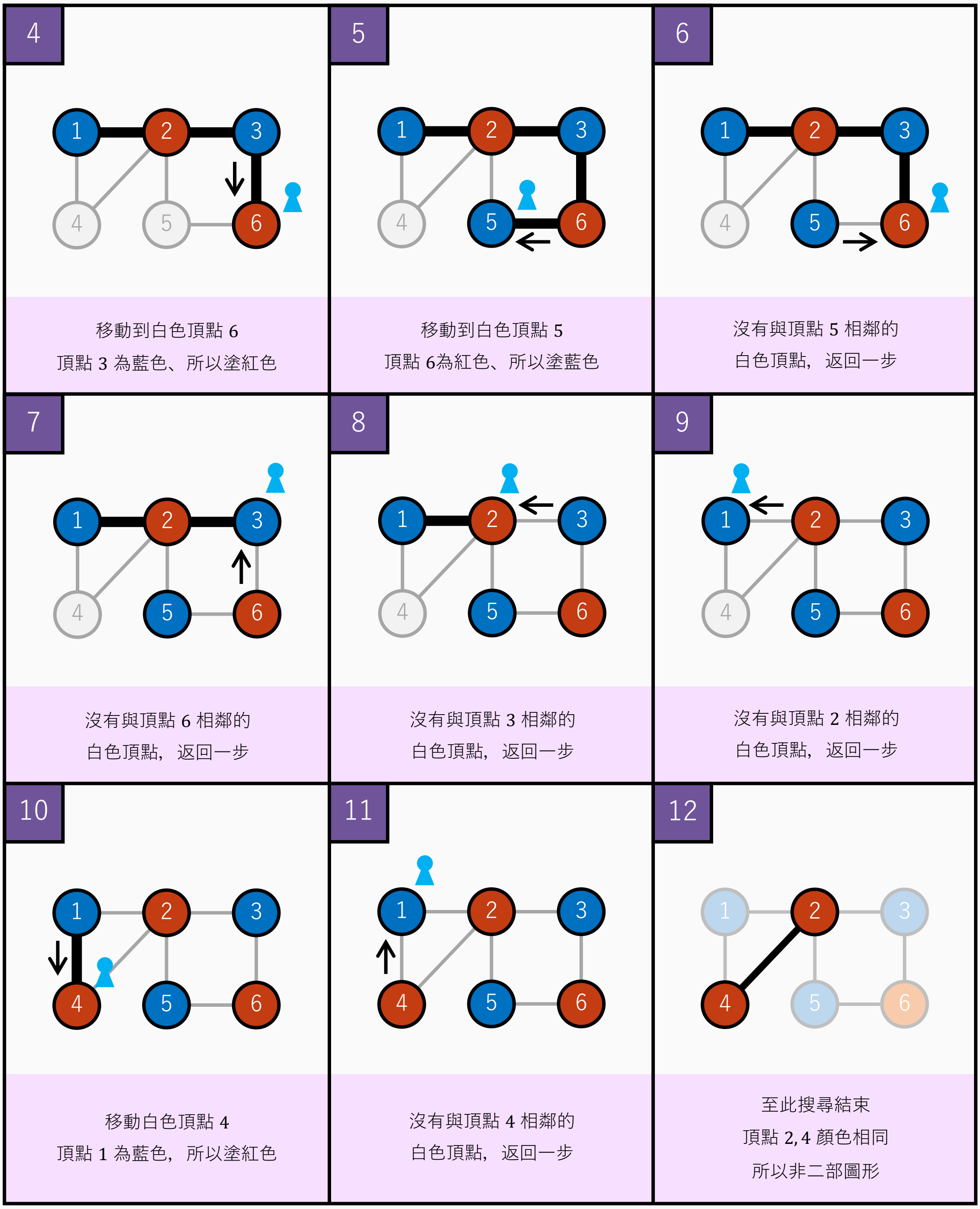


因此，進行以下的深度優先搜尋的話，可以構成一種將圖形用藍色和紅色著色的方法。另外，用藍色文字表示的部分是與4.5.6項所述的連通判斷演算法不同的部分。

1. 將所有頂點塗為白色。
2. 一開始訪問頂點 1，並將頂點 1 塗為**藍色**。
3. 之後，重複以下操作：
    A）沒有相鄰的白色頂點：返回一步
    B）有相鄰的白色頂點：訪問其中編號最小的頂點。在訪問新的頂點時，用**與前一頂點不同的顏色**著色。
4. 最終，**如果任 2 個相鄰的頂點都以不同顏色著色的話，該圖形為二部圖形る**。

將此演算法應用於具體的圖形如下。粗線表示移動路徑。





實作這個演算法如次頁。由於在程式中無法真的將頂點塗為白、藍、紅色，因此設定如下。

- `color[i]=0` 時：頂點 i 為白色
- `color[i]=1` 時：頂點 i 為藍色
- `color[i]=2` 時：頂點 i 為紅色

此外，由於也有輸入的案例為非連通圖形的可能性，注意需要對各連通分量進行相當於步驟2.的操作（參照程式中的「深度優先搜尋」部分）。

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

int N, M, A[200009], B[200009]; // N, M ≦ 200000 , 因此使陣列的大小為 200009
vector<int> G[200009];
int color[200009];

void dfs(int pos) {
    for (int i : G[pos]) { // 範圍 for 敘述
        if (color[i] == 0) {
            // color[pos]=1 的時候為 2, color[pos] = 2 的時候為 1
            color[i] = 3 - color[pos];
            dfs(i);
        }
    }
}

int main() {
    // 輸入
    cin >> N >> M;
    for (int i = 1; i <= M; i++) {
        cin >> A[i] >> B[i];
        G[A[i]].push_back(B[i]);
        G[B[i]].push_back(A[i]);
    }

    // 深度優先搜尋
    for (int i = 1; i <= N; i++) color[i] = 0;
    for (int i = 1; i <= N; i++) {
        if (color[i] == 0) {
            // 頂點 i 為白（尚未搜尋的連通成分）
            color[i] = 1;
            dfs(i);
        }
    }

    // 是否為二部圖的判斷
    bool Answer = true;
    for (int i = 1; i <= M; i++) {
        if (color[A[i]] == color[B[i]]) Answer = false;
    }
    if (Answer == true) cout << "Yes" << endl;
    else cout << "No" << endl;
    return 0;
}
```

※ Python等原始碼請參閱chap4-5.md。

問題 4.5.8

注意：此問題使用4.5.8項「其他代表性的圖演算法」介紹的Dijkstra演算法。初學者無法解決是自然的，不必擔心。

一般來說，整數可以透過重複進行「乘以 10再加上 1 至 9 的值」的操作來產生。例如，整數 8691 為：

- 從整數 0 開始
- 乘以 10 再加上 8 （$0\times10+8=8$ になる）
- 乘以 10 再加上 6 （$8\times10+6=86$ になる）
- 乘以 10 再加上 9 （$86\times10+9=869$ になる）
- 乘以 10 再加上 1 （$869\times10+1=8691$ になる）

因此，考慮如下的加權有向圖。頂點 i $(0\le i<K)$ 表示「除以 K 且餘數為 i 的整數。

關於頂點

- 準備 K 個頂點。
- 令頂點編號分別為 $0,1,2,3,\dots,K-1$ 。

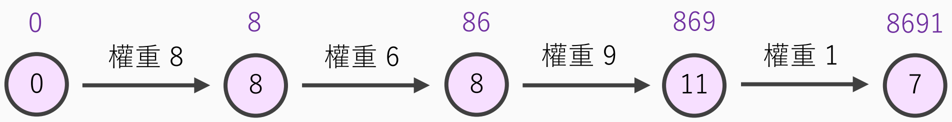
關於邊

對各頂點 i $(0\le i<K)$，添加以下 10 條邊：

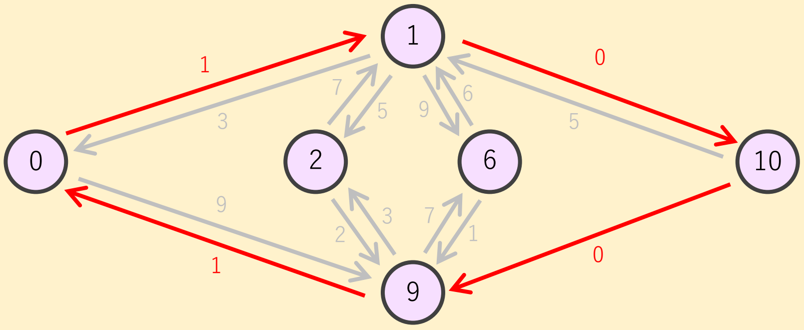
- 從頂點 i 到頂點 $(10i+0) \bmod K$ 的權重為 0 的邊※
- 從頂點 i 到頂點 $(10i+1) \bmod K$ 的權重為 1 的邊
- 從頂點 i 到頂點 $(10i+2) \bmod K$ 的權重為 2 的邊
- 從頂點 i 到頂點 $(10i+3) \bmod K$ 的權重為 3 的邊
- 從頂點 i 到頂點 $(10i+4) \bmod K$ 的權重為 4 的邊
- 從頂點 i 到頂點 $(10i+5) \bmod K$ 的權重為 5 的邊
- 從頂點 i 到頂點 $(10i+6) \bmod K$ 的權重為 6 的邊
- 從頂點 i 到頂點 $(10i+7) \bmod K$ 的權重為 7 的邊
- 從頂點 i 到頂點 $(10i+8) \bmod K$ 的權重為 8 的邊
- 從頂點 i 到頂點 $(10i+9) \bmod K$ 的權重為 9 的邊

※ 為了實作的方便，例外地不添加頂點 0 → 0的邊。

在此，通過一條邊對應於一次操作。例如，$K = 13$ 時將 86 變為 869 的操作是對應於通過從頂點 8 → 頂點 11 的權重 9 的邊（$86 \bmod 13 = 8$，$869 \bmod 13 = 11$）。



因此，從頂點 0 到頂點 $i\ (\geq 1)$ 的最短路徑長度為除以 K 餘 i 的數之中，各個位數之和的最小值。
同樣的，從頂點 0 回到頂點 0 的最短路徑長度為 K 的倍數中，各個位數之和的最小值。具體例如下。

> 例如當 $K = 13$ 時，各個位數之和的最小值是 2（數 1001），對應的路徑是 $0 \to 1 \to 10 \to 9 \to 0$（補充：$1 \bmod 13 = 1$，$10 \bmod 13 = 10$，$100 \bmod 13 = 9$、$1001 \bmod 13 = 0$）。
>
> 這條路徑是按照前一頁的方法所構成的圖形中的最短路徑。（為了方便閱讀，省略了一些頂點和邊）
>
> 

因此，使用Dijkstra算法（→**4.5.8項**）求出加權圖的最短路徑長度即可得到正確答案。

但是，直接實作時從頂點 0 到 0 的最短路徑長度會變為 0 因此需要做一些處理。具體參考次頁的實作例。

※ 也可以參考更詳細的官方解說（chap4-5.md 中有連結）。

```cpp
#include <bits/stdc++.h>
using namespace std;

int K, dist[100009];
bool used[100009];
vector<pair<int, int>> G[100009];
priority_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> Q;

// Dijkstra法
void dijkstra() {
    // 陣列的初始化等
    for (int i = 0; i < K; i++) dist[i] = (1 << 30);
    for (int i = 0; i < K; i++) used[i] = false;
    Q.push(make_pair(0, 0)); // 注意此處勿使 dist[0] = 0 ！

    // 佇列更新
    while (!Q.empty()) {
        int pos = Q.top().second; Q.pop();
        if (used[pos] == true) continue;
        used[pos] = true;
        for (pair<int, int> i : G[pos]) {
            int to = i.first, cost = dist[pos] + i.second;
            if (pos == 0) cost = i.second; // 頂點 0 時為例外}
            if (dist[to] > cost) {
                dist[to] = cost;
                Q.push(make_pair(dist[to], to));
            }
        }
    }
}

int main() {
    // 輸入
    cin >> K;

    // 添加圖形的邊
    for (int i = 0; i < K; i++) {
        for (int j = 0; j < 10; j++) {
            if (i == 0 && j == 0) continue;
            G[i].push_back(make_pair((i * 10 + j) % K, j));
        }
    }

    // Dijkstra法、輸出
    dijkstra();
    cout << dist[0] << endl;
    return 0;
}
```

※ Python等原始碼請參閱 chap4-5.md。