# 基本解法

## Java代码

```
    public int minDominoRotations(int[] A, int[] B) {
        int n = A.length;
        //记录1-6数字分别在A，B中出现的次数以及在多米诺骨牌出现的次数
        int[] countA = new int[6];
        int[] countB = new int[6];
        int[] countAB = new int[6];
        for (int i=0; i<n; i++){
            countA[A[i]-1]++;
            countB[B[i]-1]++;
            countAB[A[i]-1]++;
            if(A[i] != B[i])
                countAB[B[i]-1]++;
        }
        //在所有在所有多米诺骨牌中出现的数字中寻找最小多米诺旋转次数
        int minRotations=-1;
        for(int i=0; i<6; i++){
            if(countAB[i] == n) {
                minRotations = n - Math.max(countA[i], countB[i]);
               break;
            }
        }
        return minRotations;
    }
```

# 优化解法

## Java代码

```
public int minDominoRotations(int[] A, int[] B) {
    int n = A.length;
    //求解A或B全部变成A[0]，最少需要多少次旋转
    int rotations=check(A[0],A,B,n);
    //如果A[0]==B[0]，那么不用继续检查B[0]
    //如果A[0]!=B[0] 且可以将A或B中的元素全部变成A[0]，那么也不用再检查B[0]
    if(rotations!=-1||A[0]==B[0]){
        return rotations;
    }else {
        //如果A[0]不满足并且A[0]!=B[0]
        // 求解A或B全部变成B[0]，最少需要多少次旋转
        return check(B[0],A,B,n);
```

```
    }
}

//检查将A或者B中元素全部变成x需要多少次旋转
public int check(int x, int[] A, int[] B, int n) {
    //rotationsA存储将A中元素变成x需要多少次旋转，rotationsB存储将B中元素变成x需要多少次旋转
    int rotationsA=0,rotationsB=0;
    //遍历骨牌判断是否能完成任务（在A中完成或者在B中完成）
    for(int i=0;i<n;i++) {
        // 如果当前多米诺骨牌上没有数字x，那么不可能完成任务
        if(A[i]!=x&&B[i]!=x){
            return -1;
        }else if(A[i]!=x){
            // 如果当前多米诺骨牌上A[i]不是x，那么rotationsA需要+1
            rotationsA++;
        }else if(B[i]!=x){
            // 如果当前多米诺骨牌上B[i]不是x，那么rotationsB需要+1
            rotationsB++;
        }
    }
    // 返回最小旋转次数
    return Math.min(rotationsA,rotationsB);
}
```

# C++代码

```
class Solution {
    public:
    int check(int x, vector<int>& A, vector<int>& B, int n) {
        //检查将A或者B中元素全部变成x需要多少次旋转
        //rotations_a存储将A中元素变成x需要多少次旋转
        //rotations_b存储将B中元素变成x需要多少次旋转
        int rotations_a = 0, rotations_b = 0;
        for (int i = 0; i < n; i++) {
            // 如果当前多米诺骨牌上没有数字x，那么不可能完成任务
            if (A[i] != x && B[i] != x) return -1;
            // 如果当前多米诺骨牌上A[i]不是x，那么如果要将A中元素全部变成x需要旋转一次
            else if (A[i] != x) rotations_a++;
            // 如果当前多米诺骨牌上B[i]不是x，那么如果要将B中元素全部变成x需要旋转一次
            else if (B[i] != x) rotations_b++;
        }
        // 返回最小旋转次数
        return min(rotations_a, rotations_b);
    }

    int minDominoRotations(vector<int>& A, vector<int>& B) {
        int n = A.size();
```

```
        //检查如果将A或B中的元素全部变成A[0]，最少需要多少次旋转
        int rotations = check(A[0], B, A, n);
        //如果A[0]==B[0]，那么不用继续检查B[0]
        //如果A[0]!=B[0] 且可以将A或B中的元素全部变成A[0]，那么也不用再检查B[0]
        //因为即使可以将A或B中的元素全部变成B[0]，需要的最小旋转次数也会和A[0]一样
        if (rotations != -1 || A[0] == B[0]) return rotations;
        else return check(B[0], B, A, n);
    }
};
```

# Python代码

```
class Solution(object):
    def minDominoRotations(self, A, B):
        """
        :type A: List[int]
        :type B: List[int]
        :rtype: int
        """
        def check(x):
            #检查将A或者B中元素全部变成x需要多少次旋转
            #rotations_a存储将A中元素变成x需要多少次旋转
            #rotations_b存储将B中元素变成x需要多少次旋转
            rotations_a = rotations_b = 0
            for i in range(n):
                #如果当前多米诺骨牌上没有数字x，那么不可能完成任务
                if A[i] != x and B[i] != x:
                    return -1
                #如果当前多米诺骨牌上A[i]不是x，那么如果要将A中元素全部变成x需要旋转一次
                elif A[i] != x:
                    rotations_a += 1
                #如果当前多米诺骨牌上B[i]不是x，那么如果要将B中元素全部变成x需要旋转一次
                elif B[i] != x:
                    rotations_b += 1
            #返回最小旋转次数
            return min(rotations_a, rotations_b)

        n = len(A)
        #检查如果将A或B中的元素全部变成A[0]，最少需要多少次旋转
        rotations = check(A[0])
        #如果A[0]==B[0]，那么不用继续检查B[0]
        #如果A[0]!=B[0] 且可以将A或B中的元素全部变成A[0]，那么也不用再检查B[0]
        #因为即使可以将A或B中的元素全部变成B[0]，需要的最小旋转次数也会和A[0]一样
        if rotations != -1 or A[0] == B[0]:
            return rotations
        else:
            return check(B[0])
```