TA-Naive-Really-Naive ver.

想法

1. 我們只會從 n_K 個班型中挑一個班型當作排班依據，挑選方式為挑選一個班型其包含的時段在接下來 n_J 天的人力需求量是最多的。以簡單的例子來說明，如果有三天要安排，每天的時段只有 1,2,3 三種；班型有三種，分別為時段【$1,2$】、時段【$1,3$】及時段【$2,3$】；每天每時段的人力需求量如下表所示：

第 n 天 \ 時段	1	2	3
1	15	36	14
2	2	26	20
3	40	48	4

   則挑選最適班型的方式就是去找一個班型，其包含的時段在接下來三天的人力需求量是最多的，因此時段【$1,2$】的需求量為 $(15+36)+(2+26)+(40+48)=167$；時段【$1,3$】的需求量為 $(15+14)+(2+20)+(40+4)=95$；時段【$2,3$】的需求量為 $(36+14)+(26+20)+(48+4)=148$。因此我們會選擇時段【$1,2$】當作最好的班型。

2. 根據第一步，我們能夠求出一個最適班型。而每天該班型的需求量就是當天各時段人力需求量的平均值 (無條件捨去成整數)。同樣以上一步的例子說明，挑選出時段【$1,2$】為最佳班型後，我們就要決定這個班型在接下來 3 天每天的班型需求量。其第一天的需求量就是第一天人力需求量加總後的平均，也就是 $\frac{15+36+14}{3}=21$；第二天為 $\frac{2+26+20}{3}=16$；第三天為 $\frac{40+48+4}{3}=30$。

3. 接下來，我們就開始對每一天的每個員工進行排班，只要班型需求還沒滿足且這位員工還可以工作，就讓他做上一步挑出來的班型 (best index)；如果他因為硬性休息規定不能工作就讓他休息。

Pseudo Code

```
int main()
{
    // shiftTime 為 given input，代表每個班型的時段
    // workerDemand 為 given input，每一列代表該天各時段的工作需求量
    // 準備一個 1d array int[n_J] shiftDemand 代表每天各時段的平均需求量，讓函數能夠更新其數值
    int[n_J] shiftDemand = {0};
    int bestShiftIndex = CalculateShiftDemandAndUpdateNaive(shiftTime, workerDemand, shiftDemand);

    // 準備一個 1d array int workDays[n_J]，用來紀錄每一個員工已經工作的天數 (不能工作超過 n_I - L 天)
    // 準備一個 2d array int workSschedule[n_I][n_J]，每一個 row 代表那一名員工在接下來每一天的班型 (all initialized to -1 代表沒被指派班型)

    for (每一天 j)
    {
        int demand = shiftDemand[j]; // demand 代表當天各時段的平均需求量，也就是我們要盡量找員工滿足的需求量
        // 接下來會 traverse all employees, 只要
        // 1. 目前還有需求 (demand > 0)
        // 2. 員工還可以工作 (當天還沒被指派工作 & 工作天數還沒超過 n_I - L)
        就安排他去班型 k, 然後 demand--
        for (每一名員工 i)
        {
            if (demand > 0) // 還有工作需求
            {
                // 如果員工 i 的工作天數滿了，就讓他休息並看下一個員工 (班型設為 0)
                if (workDays[i] == n_I - L)
                {
                    workSchedule[i][j] = 0;
                    continue;
                }
                // 如果員工 i 前六天都有被指派工作，就讓他休息並看下一個員工 (班型設為 0)
                if (...) // implement it yourselves
                {
                    workSchedule[i][j] = 0;
                    continue;
                }
                // 如果員工 i 第 j 天還沒被指派工作，就指派員工 i 去工作
                if (workSchedule[i][j] == -1)
                {
                    workSchedule[i][j] = bestShiftIndex;
                    workDays[i]++;
                    demand--;
                }
            }
        }
    }
    // 此時 workSchedule 就是一個合乎規範的排班表
    return 0;
}
```

CalculateShiftDemandAndUpdateNaive function implementation

```
int CalculateShiftDemandAndUpdateNaive(int[n_K][24] shiftTime, int[n_J][24]
workerDemand, int[n_J] shiftDemand)
{
    // shiftTime 為 given input，代表每個班型的時段
    // workerDemand 為 given input，每一列代表該天各時段的工作需求量
    // shiftDemand 為 output，每一列代表該天最佳班型的需求量
    // return value 為最佳班型的 index

    int[n_K] shiftWorkerDemand = {0}; // 每個班型在接下來 n_J 天能夠滿足的總需求量
    for (每一天 j)
    {
        for (每個班型 k)
        {
            for (每個時段 i)
            {
                if (shiftTime[k][i] == 1)
                {
                    shiftWorkerDemand[k] += workerDemand[j][i];
                }
            }
        }
        // 計算當天各時段的平均工作需求量，當作最佳班型在當天的需求量
        int sum = 0;
        for (每個時段 i)
        {
            sum += workerDemand[j][i];
        }
        shiftDemand[j] = sum / 24;
    }
    // 此時最佳班型的 index 為 shiftWorkerDemand 中最大值的 index
    int maxIndex = 0;
    for (每個班型 k)
    {
        if (shiftWorkerDemand[k] > shiftWorkerDemand[maxIndex])
        {
            maxIndex = k;
        }
    }
    return maxIndex;
}
```