#include <iostream>  
#include <algorithm>  
#include <cstring>  
  
using namespace std;  
  
const int MAX\_INT = 1e9; // 最大整数  
  
int n; // 工作数量  
int c[101][101]; // 成本矩阵  
  
int min\_cost = MAX\_INT; // 用于记录最小成本  
  
// 限界函数，估计剩余决策的最优解  
int bound(int depth, int cost, int assignment[]) {  
// 计算剩余的工作数  
int remain = n - depth + 1;  
// 将剩余工作按成本从小到大排序  
sort(c[depth], c[depth] + remain, [](int x, int y) { return x < y; });  
// 将剩余工作分配给成本最小的人  
for (int i = 1; i <= remain; i++) {  
cost += c[depth + i - 1][i - 1];  
}  
return cost;  
}  
  
// 递归函数  
void assign(int depth, int cost, int upper, int assignment[]) {  
// 如果当前的成本已经大于 min\_cost，则直接返回  
if (cost >= min\_cost) return;  
// 如果所有工作都已分配，更新最小成本  
if (depth > n) {  
 min\_cost = min(min\_cost, cost);  
 return;  
}  
  
// 计算限界函数的值  
int bound\_value = bound(depth, cost, assignment);  
// 如果限界函数的值已经大于 upper，则剪枝  
if (bound\_value >= upper) return;  
  
// 遍历所有人，尝试分配当前工作  
for (int i = 1; i <= n; i++) {  
 // 如果当前人已被分配过工作，则跳过  
 if (assignment[i] != 0) continue;  
  
 // 分配工作  
 assignment[i] = depth;  
 // 递归搜索  
 assign(depth + 1, cost + c[i][depth], upper, assignment);  
 // 撤销分配  
 assignment[i] = 0;  
}  
}  
  
  
  
  
  
  
int main() {  
cin >> n;  
for (int i = 1; i <= n; i++) {  
for (int j = 1; j <= n; j++) {  
cin >> c[i][j];  
}  
}  
int assignment[n+1]; // 用于记录每个人分配的工作  
memset(assignment, 0, sizeof(assignment)); // 初始化为未分配状态  
assign(1, 0, MAX\_INT, assignment); // 从第1个工作开始分配，初始成本为0，上界为最大整数  
cout << min\_cost << endl; // 输出最小成本  
return 0;  
}

算法思路

1. 用一个二维数组存储每个人干每项工作的成本 cn+1，ci表示第i个人干第j项工作的成本
2. 决策树的每个结点表示一个人干一项工作的情况，每个结点有n个分支，分别表示干第1项、第2项、...、第n项工作的情况
3. 上界函数 bound() 用于估计剩余决策的最优解，即剩余工作分配给成本最小的人，如果这个值已经大于当前的最优解，则剪枝

伪代码如下