题目描述

在通信系统中，一个常见的问题是对用户进行下时策略的调度，会得到不同的系统消耗和性能。

假设当前有个待串行调度用户，每个用户可以使用A/B/C三种不同的调度策略，不同的策略会消耗不同的系统资源。请你根据如下规则进行用户调度，并返回总的消耗资源量。

规则：

1. 相同的用户下能使用相同的调度策略，例如，第1个用户使用了A策略，则第2个用户只能使用B或者C策略。

2. 对单个用户而言，不同的调度策略对系统资源的消耗可以 归一化 的独立加数量。例如，某用户分别使用A/B/C策略的系统消耗分别为15/8/17。

3. 每个用户依次选择当前所能选择的对系统资源消耗最少的策略（局部最优），如果有多个满足要求的策略，选最后一个。

输入描述

第一行表示用户个数n

接下来每一行表示一个用户分别使用三个策略时系统消耗resA resB resC

输出描述

最优策略组合下的总的系统资源消耗数

用例

输入	3 15 8 17 12 20 9 11 7 5
输出	24
说明	1号用户使用B策略，2号用户使用C策略，3号用户使用B策略。系统资源消耗 8 + 9 + 7 = 24。

题目解析

这题应该是要我们使用dfs来解。

按照题目意思，用户是串行调度的，因此执行顺序不能改变，即dfs第一层必须选15 8 17，第二层必须选12 20 9，第三层必须选11 7 5，并且相邻层级间不能使用相同系统消耗来。

即

第一层选了15。

第二层就不能选12，但是可以选20或9。如果第二层选了20。

则第三层就不能选7，但是可以选11或5。

因此dfs的递归思路，每层节点的筛选条件我们就都有了。

JavaScript算法源码

```
// 注意类名必须为 Main, 不要有任何 package 语句
const readline = require("readline");

const rl = readline.createInterface({
  input: process.stdin,
  output: process.stdout,
});

const lines = [];
let m;
rl.on("line", (line) => {
  lines.push(line);

  if (lines.length === 1) {
    m = parseInt(lines[0]);
  }

  if (m != undefined && lines.length === m + 1) {
    lines.shift();
    const res = lines.map((line) => line.split(" ").map(Number));
    const ans = [Infinity];
    dfs(res, m, 0, -1, 0, ans);
    console.log(ans[0]);

    lines.length = 0;
  }
});

function dfs(res, m, level, index, total, ans) {
  if (level == m) {
    ans[0] = Math.min(ans[0], total);
    return;
  }

  const r = res[level];
  for (let i = 0; i < r.length; i++) {
    if (i != index) {
      dfs(res, m, level + 1, i, total + r[i], ans);
    }
  }
}
```

Java算法源码

```
import java.util.Scanner;

public class Main {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    int m = sc.nextInt();

    int[][] res = new int[m][3];
    for (int i = 0; i < m; i++) {
      res[i][0] = sc.nextInt();
      res[i][1] = sc.nextInt();
      res[i][2] = sc.nextInt();
    }

    int[] ans = {Integer.MAX_VALUE};
    dfs(res, m, 0, -1, 0, ans);
    System.out.println(ans[0]);
  }

  public static void dfs(int[][] res, int m, int level, int index, int total, int[] ans) {
    if (level == m) {
      ans[0] = Math.min(ans[0], total);
      return;
    }

    int[] r = res[level];
    for (int i = 0; i < r.length; i++) {
      if (i != index) {
        dfs(res, m, level + 1, i, total + r[i], ans);
      }
    }
  }
}
```

Python算法源码

```
import sys

# 输入获取
m = int(input())
res = list(map(int, input().split()) for _ in range(m))


# 算法入口
def dfs(level, index, total, ans):
    if level == m:
        ans[0] = min(ans[0], total)
        return

    r = res[level]
    for i in range(len(r)):
        if i != index:
            dfs(level + 1, i, total + r[i], ans)


# 算法调用
ans = [sys.maxsize]
dfs(0, -1, 0, ans)
```