

**《算法设计与分析实践》**

**实 验 报 告**

成

绩：

评

语：

教师签字：

评阅日期：

**专业班级： 计算机**

**学 号：**

**姓 名：**

**指导教师：**

**报告日期： 2024.12.27**

**计算机科学与技术学院**

**2024 年 11 月**

目录

[一、 实验完成情况 2](#_Toc186796838)

[二、 P1220 关路灯 解题报告 2](#_Toc186796839)

[1. 题目分析 2](#_Toc186796840)

[2. 算法分析 2](#_Toc186796841)

[3. 性能分析 2](#_Toc186796842)

[4. 运行测试 2](#_Toc186796843)

[三、 P2512糖果传递 解题报告 2](#_Toc186796844)

[1. 题目分析 2](#_Toc186796845)

[2. 算法分析 2](#_Toc186796846)

[3. 性能分析 2](#_Toc186796847)

[4. 运行测试 2](#_Toc186796848)

[四、 P1784数独 解题报告 2](#_Toc186796849)

[1. 题目描述 2](#_Toc186796850)

[2. 算法分析 2](#_Toc186796851)

[3. 性能分析 2](#_Toc186796852)

[4. 运行测试 2](#_Toc186796853)

[五、 P3368 树状数组2 解题报告 2](#_Toc186796854)

[1. 题目描述 2](#_Toc186796855)

[2. 算法分析 2](#_Toc186796856)

[3. 性能分析 2](#_Toc186796857)

[4. 运行测试 2](#_Toc186796858)

[六、 实验总结与体会 2](#_Toc186796859)

[1. 个人收获和心得 2](#_Toc186796860)

[2. 课程感想 2](#_Toc186796861)

[附录： 源代码 2](#_Toc186796862)

[1. P1220 关路灯 源代码 2](#_Toc186796863)

[2. P2512糖果传递 源代码 2](#_Toc186796864)

[3. P1784数独 源代码 2](#_Toc186796865)

[4. P3368 树状数组2 源代码 2](#_Toc186796866)

# 实验完成情况

遵循实验要求，共完成22题，各题的单元分布如下表所示。

表格 1 实验完成情况

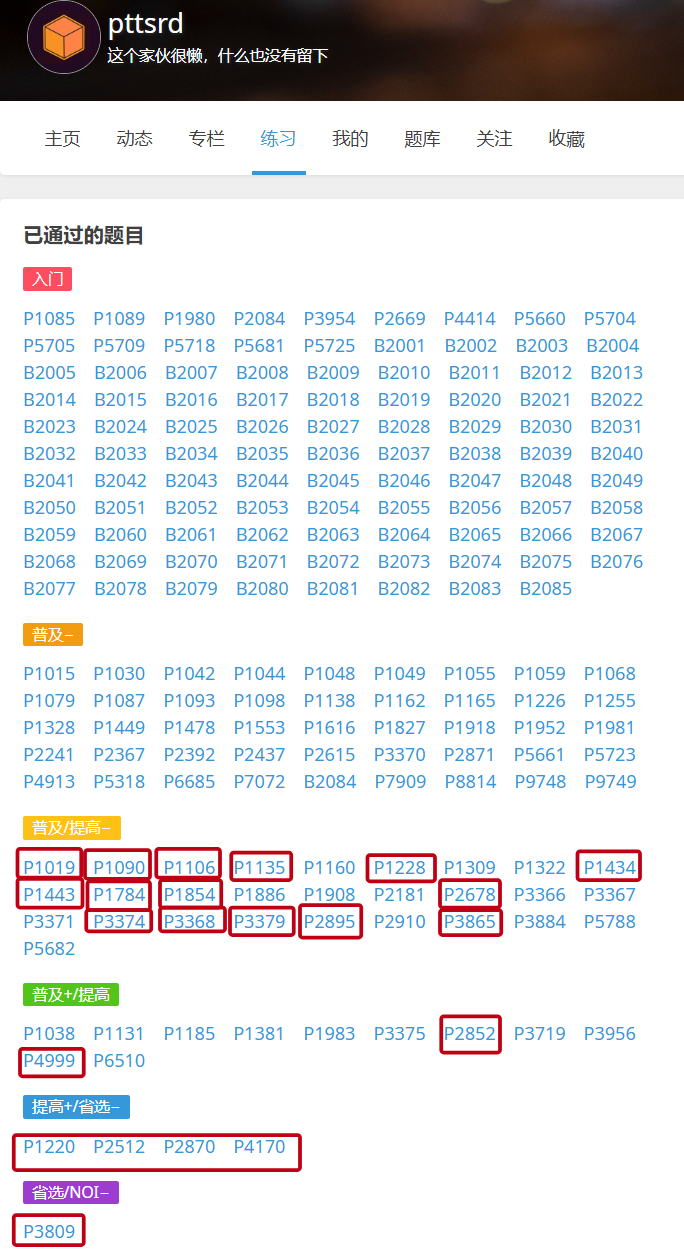
|  |  |  |  |
| --- | --- | --- | --- |
| 单元 | 实验内容 | 完成题数 | 所选题目 |
| 1 | 基于分治法的问题求解 | 2 | P2678 跳石头  P1228地毯填补问题 |
| 2 | 基于动态规划方法的问题求解 | 5 | P1220关路灯  P4170 涂色  P1854花店橱窗布置  P4999烦人的数学作业  P1434滑雪 |
| 3 | 基于贪心策略的问题求解 | 3 | P1106删数问题  P2512糖果传递  P1090合并果子 |
| 4 | 图相关问题的求解 | 3 | P1443马的遍历  P1135奇怪的电梯  P2895 Meteor Shower S |
| 5 | 搜索算法的设计与应用 | 5 | P1784数独  P1019单词接龙  P3809 后缀排序  P2852 Milk Patterns G  P2870 Best Cow Line G |
| 6 | 高级数据结构与算法专题 | 4 | P3374 树状数组 1  P3368 树状数组 2  P3379 最近公共祖先（LCA）  P3865 ST 表 && RMQ 问题 |

以上22道题目均在本地环境下运行成功并通过洛谷平台所有评测。

在洛谷网站中的完成情况截图如下：



完成人与完成状态证明见下图。



# P1220 关路灯 解题报告

## 题目分析

某一村庄在一条路线上安装了 n盏路灯，每盏灯的功率有大有小（即同一段时间内消耗的电量有多有少）。老张就住在这条路中间某一路灯旁，他有一项工作就是每天早上天亮时一盏一盏地关掉这些路灯。

为了给村里节省电费，老张记录下了每盏路灯的位置和功率，他每次关灯时也都是尽快地去关，但是老张不知道怎样去关灯才能够最节省电。他每天都是在天亮时首先关掉自己所处位置的路灯，然后可以向左也可以向右去关灯。开始他以为先算一下左边路灯的总功率再算一下右边路灯的总功率，然后选择先关掉功率大的一边，再回过头来关掉另一边的路灯，而事实并非如此，因为在关的过程中适当地调头有可能会更省一些。

现在已知老张走的速度为 1m/s，每个路灯的位置（是一个整数，即距路线起点的距离，单位：m）、功率（W），老张关灯所用的时间很短而可以忽略不计。

请你为老张编一程序来安排关灯的顺序，使从老张开始关灯时刻算起所有灯消耗电最少（灯关掉后便不再消耗电了）。

## 算法分析

本题中，我们需要求解的是老张的关灯顺序。由于老张可以向左或向右关灯，我们需要找到一种策略，使得总功耗最小。

1. **数据分析**

输入数据：

|  |  |  |
| --- | --- | --- |
| 输入 | 说明 | 数据规模 |
|  | 路灯的总数 |  |
|  | 老张所处位置的路灯号 |  |
|  | 各盏路灯的位置和功率 |  |

输出：最小功耗

1. **朴素策略**

除了最后一盏灯外，老张每关闭一盏灯，均需要做一次决策：

* 1. 继续前进
  2. 掉头后退

故对于盏灯，共需作出次决策，最坏情况下，共有种可能顺序。由于本题中，理论上可通过DFS+剪枝策略来遍历每一种可能顺序。但是，仔细分析我们不难发现本题可用动态规划的思想。

1. **动态规划**

分析问题，发现该问题满足最优子结构性质：将“剩下尚未关的灯的左右边界”和“老张当前所在的灯”当作状态。对每个状态，若决定向左关灯还是向右关灯，就进入对应的子状态，其最优解在此基础上可递归求得。这意味着从大问题分解出的子问题具有和原问题相同的形式，每个子问题的最优解能组合成最终最优解，形成最优子结构。

* 1. 状态定义：

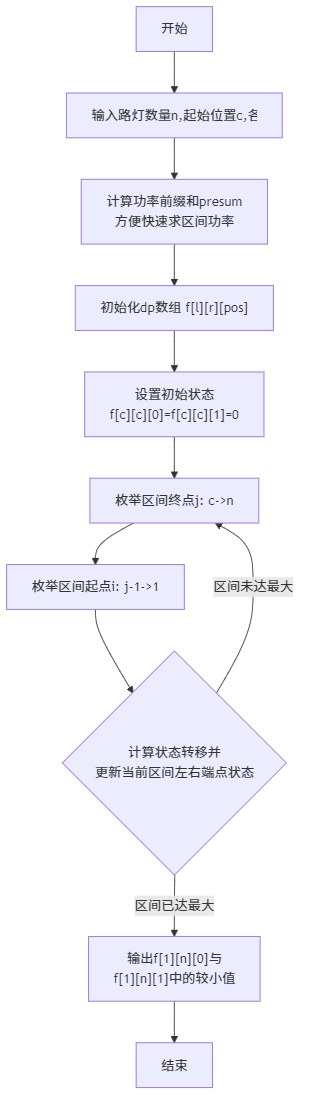
用dp[l][r][pos] 表示处理区间 [l,r] 时当前在 pos 位置的最小耗电量，pos=0 表示在左端，pos=1 表示在右端。

* 1. 状态转移：

当老张处于区间的左端时，上一个状态为区间的左端，或者区间的右端；

当老张处于区间的右端时，上一个状态为区间的左端，或者区间的右端；

算法流程图如下



算法伪代码如下：

|  |
| --- |
|  |

1. **细节优化**

使用**前缀和**来快速求区间功率：

区间关闭时的耗电量

= (剩余灯功率和+ 剩余灯功率和× 移动耗时

= × 移动耗时。

## 性能分析

预处理前缀和数组部分，时间复杂度为；区间DP部分，时间复杂度为。故该算法的总时间复杂度为。

## 运行测试

在时间限制1.00s，内存限制512.00MB下，测试全部通过。



# P2512糖果传递 解题报告

## 题目分析

有n 个小朋友坐成一圈，每人有个糖果。每人只能给左右两人传递糖果。每人每次传递一个糖果代价为 1。求使所有人获得均等糖果的最小代价。

## 算法分析

1. **数据分析**

输入数据：

|  |  |  |
| --- | --- | --- |
| 输入 | 说明 | 数据规模 |
|  | 小朋友人数 |  |
|  | 每人所持糖果数 |  |

输出：使所有人获得均等糖果的最小代价

1. **关系分析**

已知为每个小朋友原有的糖果数，为最终所持糖果数，则为的平均数。

我们用表示第个小朋友向左传递的糖果数量，为正数表示给，为负数表示拿。那么，我们的目的是最小化。

此时，我们可以列出以下关系式：

观察发现，n个方程两边相加会消掉所有的x成为一个恒等式，说明只有个方程是独立的。

用表示其他的有：

令

,

则

。

故

即求得此关于的多项式最小值。

令

对求导得

其中

当最小时，。因此各项中1与-1的数量应相同。则应为的中位数。

即

最后将代入即可得最小代价。

1. **代码优化：中位数求解**

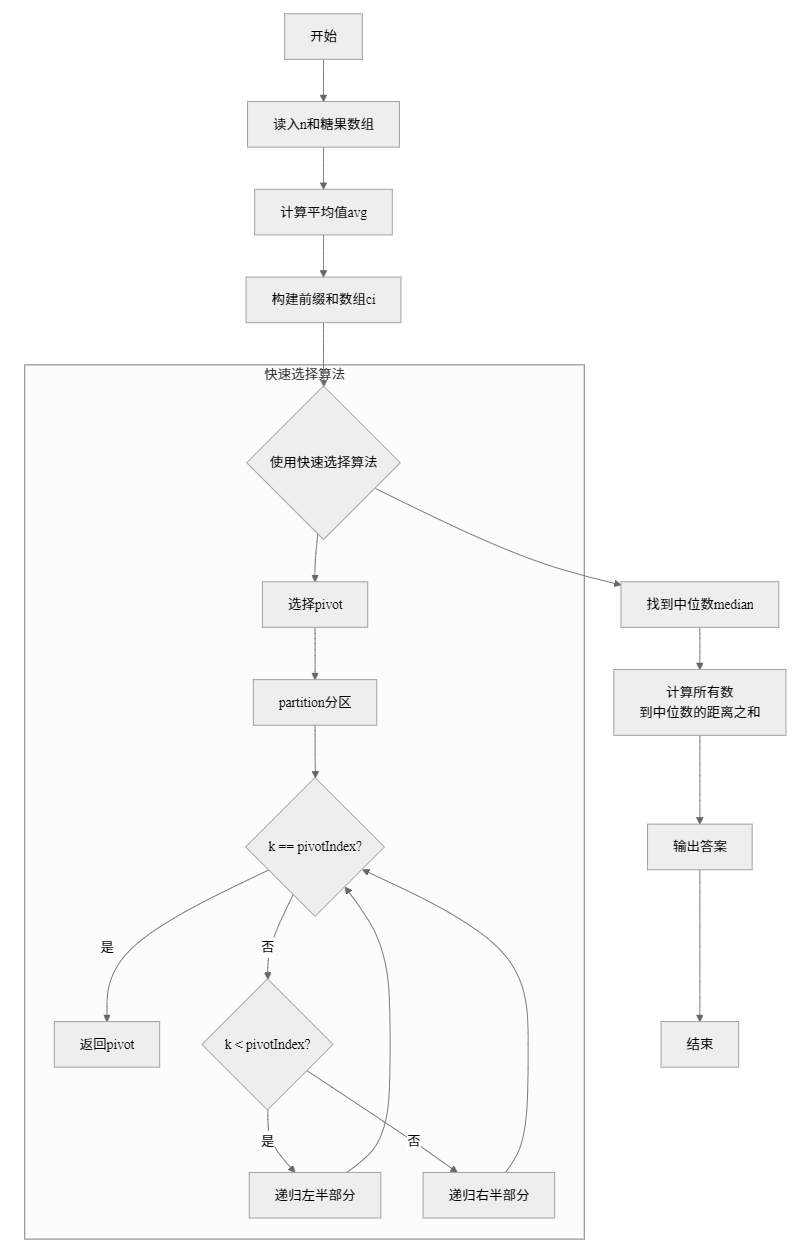
求解中位数,第一思路是使用**快速排序**算法，时间复杂度。对于本题数据规模来说不够高效。

更优的方法是采用**快速选择**算法：该算法伪代码如下：

|  |
| --- |
|  |

上述伪代码中quickselect为主函数，接受数组 A，左右边界 left 和 right，以及要找的第 k 小的元素的索引。其中用到辅助函数randompartition，随机选择一个基准元素并将数组重新排列，使得所有小于基准的元素在左边，大于基准的元素在右边。该算法的时间复杂度为

**算法总流程图：**



## 性能分析

1. **时间复杂度分析**：

读入数据:

计算前缀和:

排序找中位数:

计算代价:

总体时间复杂度:

1. **空间复杂度分析**：

存储输入数组：

排序使用的额外空间：

总空间复杂度：

该算法具有线性复杂度。由于任何算法必须访问每个小朋友的持有数，故算法复杂度至少为。因此，本算法从性能上看可以说是十分优秀的。

## 运行测试

在时间限制1.00s，内存限制512.00MB下，测试全部通过。



# P1784数独 解题报告

## 题目描述

数独是根据盘面上的已知数字，推理出所有剩余空格的数字，并满足每一行、每一列、每一个粗线宫内的数字均含，不重复。每一道合格的数独谜题都有且仅有唯一答案，推理方法也以此为基础，任何无解或多解的题目都是不合格的。

芬兰一位数学家号称设计出全球最难的“数独游戏”，并刊登在报纸上，让大家去挑战。

这位数学家说，他相信只有“智慧最顶尖”的人才有可能破解这个“数独之谜”。

据介绍，目前数独游戏的难度的等级有一到五级，一是入门等级，五则比较难。不过这位数学家说，他所设计的数独游戏难度等级是十一，可以说是所以数独游戏中，难度最高的等级。他还表示，他目前还没遇到解不出来的数独游戏，因此他认为“最具挑战性”的数独游戏并没有出现。

## 算法分析

每个空格均有9种填法，故数独填充是一颗九叉数，暴力求解的时间复杂度为，为未填充格数。不难想到回溯法（Backtracking）实现。使用深度优先搜索（DFS）策略来填充空格。通过在每一步中尝试填入一个可能的数字，并递归地继续解决剩余的部分，直到找到一个有效的解。为了提高效率，可以结合剪枝技术，以避免不必要的计算。

1. **深度优先搜索策略**

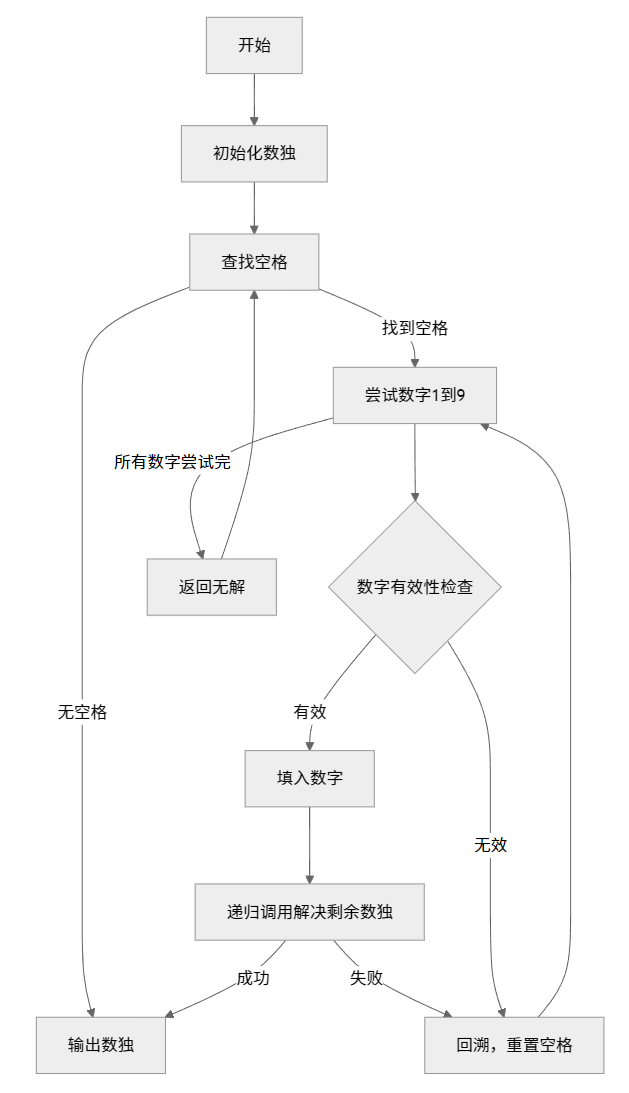
数独问题存在以下约束条件：

1. 行约束：当前行不能存在相同数字。
2. 列约束：当前列不能存在相同数字。
3. 宫约束：对应3×3小宫内不能存在相同数字。

在以上约束条件下，执行以下搜索与剪枝策略：

1. 若满足以上约束条件，则可以递归继续填下一个空格，否则立即剪枝回溯。
2. 若填入的数字使后续某格无可用数字，则在此分支回溯重试其他数字。
3. 若填满所有空格，则返回结果。

算法流程图如下：



1. **优化细节**

维护三类布尔表，用于检查行、列、3×3宫是否已填入当前数字。

行标记数组: 表示第i行是否使用了数字

列标记数组: 表示第i列是否使用了数字

宫标记数组: 表示第i个3x3宫是否使用了数字

则是否满足约束的条件判断为：

算法伪代码如下：

|  |
| --- |
|  |

## 性能分析

1. **时间复杂度分析**：

读入数据:

最坏情况:, n为空格数量

实际情况: 远好于最坏情况，因为剪枝显著减少搜索分支

1. **空间复杂度分析**：

棋盘存储: O(81) = O(1)

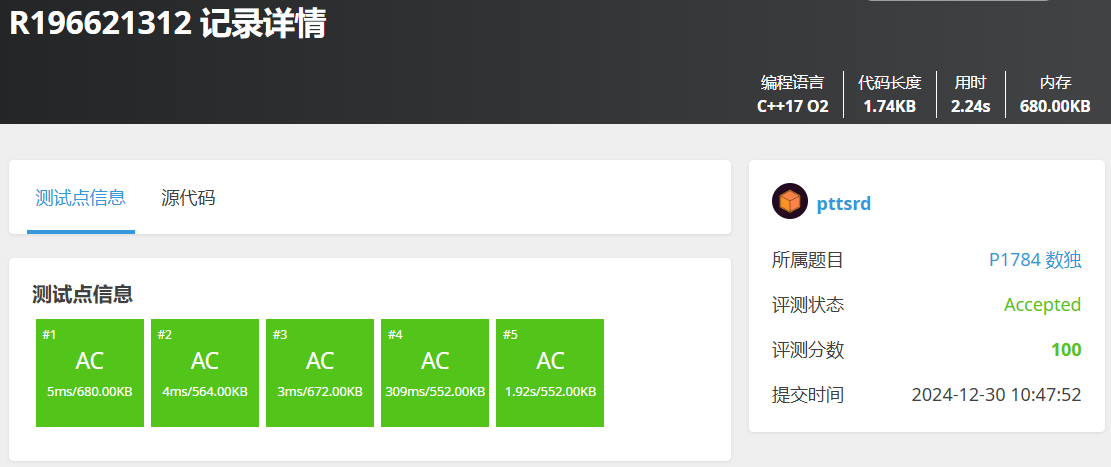
标记数组: O(9×10×3) = O(1)

递归栈深度: O(81) = O(1)

总体: O(1)

## 运行测试

在时间限制1.00s，内存限制512.00MB下，测试全部通过。



# P3368 树状数组2 解题报告

## 题目描述

已知一个数列，你需要进行下面两种操作：

1. 将某区间每一个数加上 *x*；
2. 求出某一个数的值。

**输入格式**

第一行包含两个整数 *N*、*M*，分别表示该数列数字的个数和操作的总个数。

第二行包含 *N* 个用空格分隔的整数，其中第 *i* 个数字表示数列第 *i* 项的初始值。

接下来 *M* 行每行包含 2 或 4个整数，表示一个操作，具体如下：

操作 1： 格式：1 x y k 含义：将区间  [*x*,*y*] 内每个数加上 k*k*；

操作 2： 格式：2 x 含义：输出第 *x* 个数的值。

**输出格式**

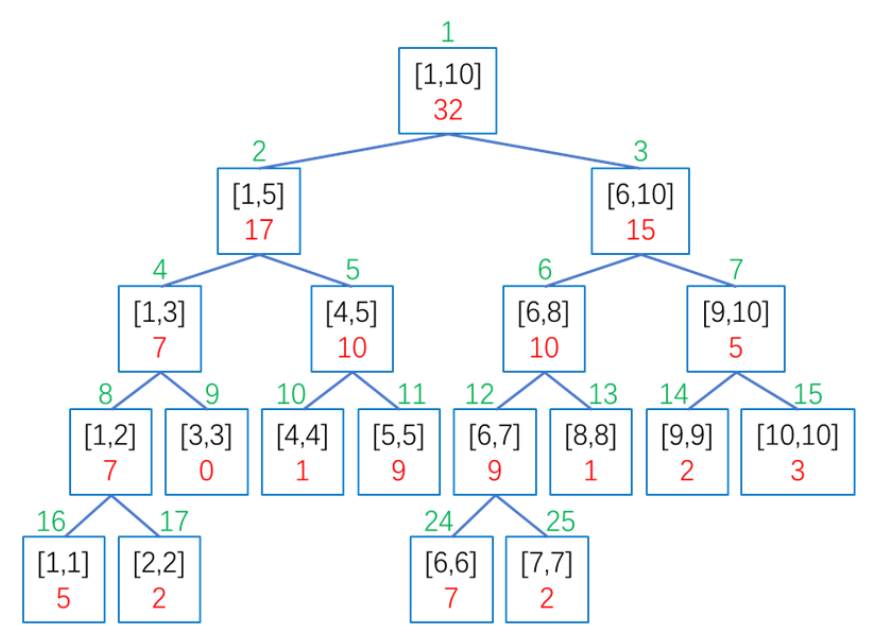
输出包含若干行整数，即为所有操作 2 的结果。

## 算法分析

由于数据较大，若采用差分思路，每次查询都需要的时间复杂度，总共需的时间复杂度，速度较慢！

1. **线段树**

本题是一道经典的线段树模板题。线段树是基于**分治思想**的二叉树，用来维护区间信息，可以在的时间内执行区间修改和区间查询。如以下示意图所示，线段树中每个叶子节点存储元素本身，非叶子节点存储区间信息。



本题中，

1. 需要支持区间修改：给区间 [l,r] 的每个数加上k
2. 需要支持单点查询：查询第x个位置的值
3. 数据范围较大 (N,M ≤ 500000)

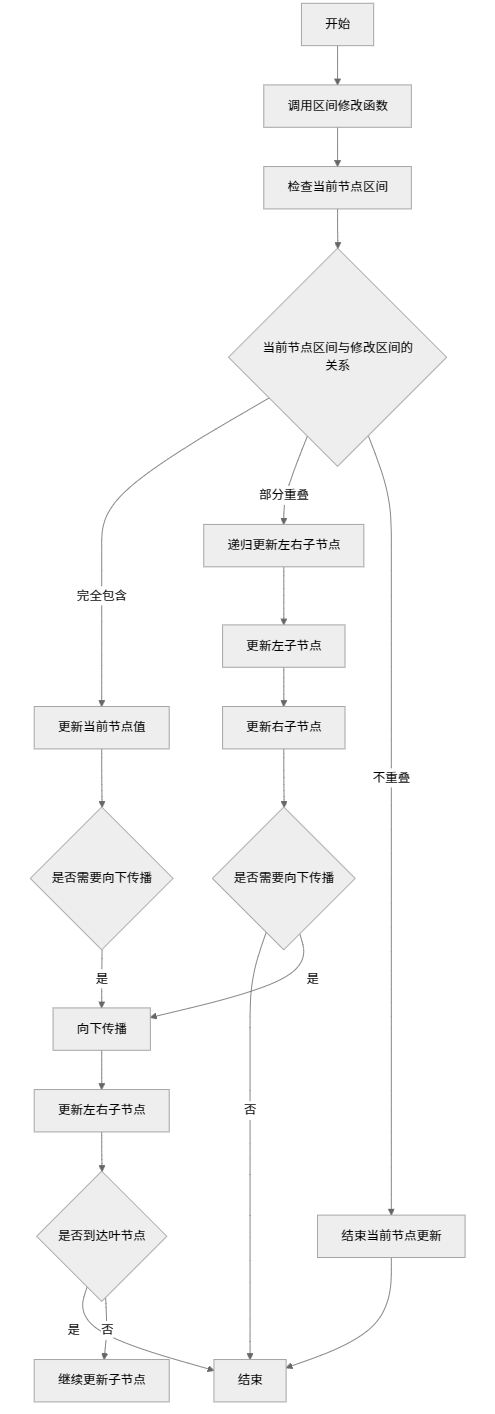
对于本题，需要做到区间维护与单点查询。

1. **线段树的区间维护**

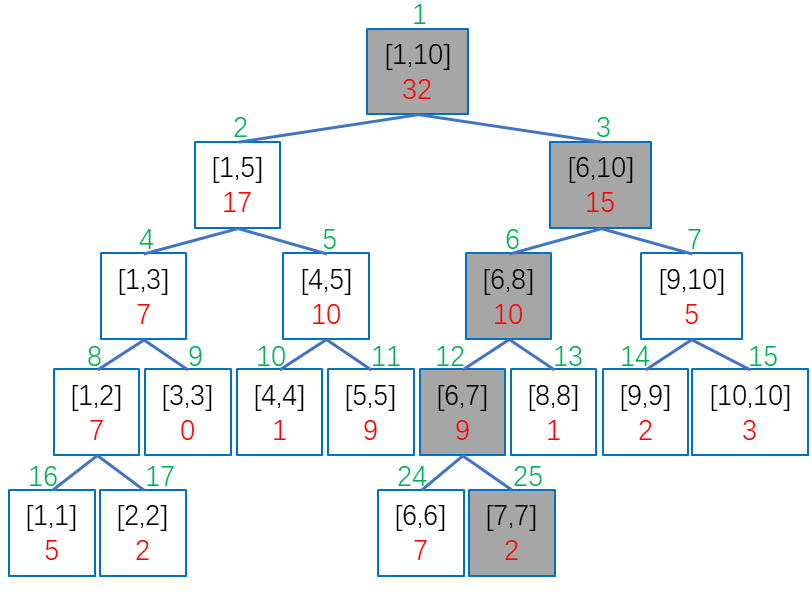
基本步骤：检查当前节点区间，确认当前节点的区间与待修改区间的关系。

* + 1. 完全包含：如果当前节点的区间完全包含待修改区间，则更新当前节点值，并决定是否向下传播。
    2. 部分重叠：如果当前节点的区间与待修改区间部分重叠，则递归更新左右子节点。
    3. 不重叠：如果当前节点的区间与待修改区间不重叠，则结束当前节点的更新。

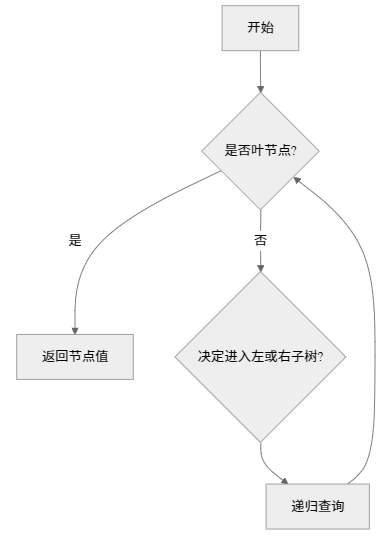
线段树区间维护的流程图如下图所示：

**

1. **线段树的单点查询**

****

线段树的单点查询流程图如下图所示：



1. **用数组存储线段树的空间大小分析**

在最理想的情况下，线段树为完全满二叉树，，则非叶子节点有个，节点的最大编号为。

一般情况下：

1. 若，则最大满层有个节点，其上共有个节点，则节点最大编号为：
2. 若，则节点最大编号为：
3. 若，则节点最大编号为：

故使用数组存储线段数结构时，最坏情况下需要 4n 的空间。

## 性能分析

1. **时间复杂度分析**：

读入数据:

线段树构建：

区间维护：

单点查询：

总计：

1. **空间复杂度分析**：

本题使用数组存储线段数结构，最坏情况下需要 4n 的空间，空间复杂度。

## 运行测试

在时间限制1.00s，内存限制512.00MB下，测试全部通过。



# 实验总结与体会

## 个人收获和心得

本次算法设计与分析实践课，让我鼓起了勇气面对曾经叹而观止的算法难题，迈出了克服恐惧心理的最重要的一步。回顾本学期算法课与上学期数据结构课，算法问题的解决离不开良好的理论知识，掌握理论知识才能在面对问题时拥有直觉与思路，才能系统性的分析设计优秀的算法。否则，只能像无头苍蝇一样，在知识碎片中迷失。对此我颇感遗憾，未能及早遇到这门课。

当然，知识体系的形成并非一门课可以促成，需要的是系统的培养体系。虽说计算机科学是最适合自学的学科，但若没有完备的培养体系，很难去精通这门学科。因此，对于那些中学时代便解除算法的同学们，我深感羡慕。

## 课程感想

本实验课设计十分完备，虽然今年是第一次在洛谷平台上进行，但有详细的任务书，题目选取与教材紧密结合，可以看到课程组的精心准备。实验课老师们经常激烈讨论，课程氛围十分融洽！

唯一美中不足的是课程开设时间较为靠后，希望以后能再提前一些！如果可以跟数据结构课程合并再好不过。

# 附录： 源代码

本地配置环境：

|  |  |
| --- | --- |
| 处理器 | 12th Gen Intel(R) Core(TM) i5-12500H 2.50 GHz |
| 系统类型 | 64-bit operating system, x64-based processor |
| 编辑器 | VScode 1.96.0 |
| 编译器 | g++.exe (x86\_64-posix-sjlj-rev0, Built by MinGW-W64 project) 8.1.0 |
| 调试器 | GNU gdb (GDB) 8.1 |
| CMAKE | cmake version 3.30.1 |
| Windows版本 | Edition Windows 11 Pro  Version 23H2  OS build 22631.4602 |

## P1220 关路灯 源代码

#include <bits/stdc++.h>

using namespace std;

const int INF = 0x3f3f3f3f;

int n, c;

int p[55], w[55], presum[55];

int f[55][55][2];

int main(int argc, char const \*argv[])

{

ios::sync\_with\_stdio(false);

cin.tie(0);

cin >> n >> c;

for (int i = 1; i <= n; i++) {

cin >> p[i] >> w[i];

presum[i] = w[i] + presum[i - 1];

}

memset(f, INF, sizeof(f));

f[c][c][0] = f[c][c][1] = 0;

for (int j = c; j <= n; j++) {

for (int i = j - 1; i >= 1; i--) {

f[i][j][0] = min(f[i + 1][j][0] + (p[i + 1] - p[i]) \* (presum[n] - presum[j] + presum[i]),

f[i + 1][j][1] + (p[j] - p[i]) \* (presum[n] - presum[j] + presum[i]));

f[i][j][1] = min(f[i][j - 1][0] + (p[j] - p[i]) \* (presum[n] - presum[j - 1] + presum[i - 1]),

f[i][j - 1][1] + (p[j] - p[j - 1]) \* (presum[n] - presum[j - 1] + presum[i - 1]));

}

}

cout << min(f[1][n][0], f[1][n][1]);

return 0;

}

## P2512糖果传递 源代码

#include <bits/stdc++.h>

#include <algorithm>

#include <cmath>

using namespace std;

#define ll long long

const int MAXN = 1e6 + 5;

int n;

ll sum, avg;

ll a[MAXN], b[MAXN];

int main(int argc, char const \*argv[])

{

ios::sync\_with\_stdio(false);

cin.tie(0);

cin >> n;

for (int i = 1; i <= n; i++) {

cin >> a[i];

sum += a[i];

}

avg = sum / n;

ll tmp = 0;

for (int i = 1; i <= n - 1; i++) {

tmp += a[i + 1];

b[i] = avg \* i - tmp;

}

sort(b, b + n);

ll m = b[(n - 1) / 2];

ll ret = 0;

for (int i = 0; i < n; i++) {

ret += abs(b[i] - m);

}

cout << ret;

return 0;

}

## P1784数独 源代码

#include <bits/stdc++.h>

using namespace std;

int board[16][16];

bool row[16][16], col[16][16], box[16][16];

void printBoard()

{

for (int i = 1; i <= 9; i++) {

for (int j = 1; j <= 9; j++) {

cout << board[i][j] << ' ';

}

cout << endl;

}

exit(0);

}

void dfs(int x, int y)

{

if (board[x][y] == 0) {

int ibox = (x - 1) / 3 \* 3 + (y - 1) / 3 + 1;

for (int i = 1; i <= 9; i++) {

if (row[x][i] || col[y][i] || box[ibox][i]) {

continue;

} else {

board[x][y] = i;

row[x][i] = col[y][i] = box[ibox][i] = true;

if (x == 9 && y == 9) {

printBoard();

} else if (y == 9) {

dfs(x + 1, 1);

} else {

dfs(x, y + 1);

}

board[x][y] = 0;

row[x][i] = col[y][i] = box[ibox][i] = false;

}

}

} else {

if (x == 9 && y == 9) {

printBoard();

} else if (y == 9) {

dfs(x + 1, 1);

} else {

dfs(x, y + 1);

}

}

}

int main(int argc, char const \*argv[])

{

ios::sync\_with\_stdio(false);

cin.tie(0);

int tmp;

for (int i = 1; i <= 9; i++) {

for (int j = 1; j <= 9; j++) {

cin >> tmp;

board[i][j] = tmp;

if (board[i][j] != 0) {

int ibox = (i - 1) / 3 \* 3 + (j - 1) / 3 + 1;

row[i][tmp] = true;

col[j][tmp] = true;

box[ibox][tmp] = true;

}

}

}

dfs(1, 1);

return 0;

}

## P3368 树状数组2 源代码

#include <bits/stdc++.h>

using namespace std;

const int MAXN = 5e5 + 5;

int a[MAXN], b[MAXN];

int N, M;

int n;

int w[MAXN];

struct {

int l, r;

int sum, add;

} T[MAXN << 2];

void build(int p, int l, int r)

{

T[p] = {l, r, w[l], 0};

if (l == r) {

return;

}

int lc = p << 1, rc = p << 1 | 1;

int mid = (l + r) >> 1;

build(lc, l, mid);

build(rc, mid + 1, r);

T[p].sum = T[lc].sum + T[rc].sum;

}

void pushdown(int p)

{

if (T[p].add) {

int lc = p << 1, rc = p << 1 | 1;

T[lc].sum += T[p].add \* (T[lc].r - T[lc].l + 1);

T[rc].sum += T[p].add \* (T[rc].r - T[rc].l + 1);

T[lc].add += T[p].add;

T[rc].add += T[p].add;

T[p].add = 0;

}

}

void update(int p, int x, int y, int k)

{

if (x <= T[p].l && T[p].r <= y) {

T[p].sum += k \* (T[p].r - T[p].l + 1);

T[p].add += k;

return;

}

pushdown(p);

int lc = p << 1, rc = p << 1 | 1;

int mid = (T[p].l + T[p].r) >> 1;

if (x <= mid) {

update(lc, x, y, k);

}

if (y > mid) {

update(rc, x, y, k);

}

T[p].sum = T[lc].sum + T[rc].sum;

}

int query(int p, int x)

{

if (x > T[p].r || x < T[p].l) { // 无

return 0;

}

if (T[p].l == T[p].r) { // 叶

return T[p].sum;

}

pushdown(p);

return query(p << 1, x) + query(p << 1 | 1, x);

}

int main(int argc, char const \*argv[])

{

ios::sync\_with\_stdio(false);

cin.tie(0);

cin >> N >> M;

for (int i = 1; i <= N; i++) {

cin >> w[i];

}

build(1, 1, N);

int x, y, k;

for (int i = 1; i <= M; i++) {

int cmd;

cin >> cmd;

if (cmd == 1) {

cin >> x >> y >> k;

update(1, x, y, k);

} else {

cin >> x;

cout << query(1, x) << endl;

}

}

return 0;

}