

综合实验报告

课题名称： 线段树的应用

课程名称： 数据结构与算法

学 院： 计算机与信息安全学院

成 员： 1. 2200330114 丁振耿

2. 2200330126 林镔壕

3. 2200330129 刘 康

报告日期： 2023年 12 月 20 日

|  |
| --- |
| 学号：2200330113 姓名：丁振耿 |
| 个人在团队中的贡献：完成课题“生长树”的代码，以及制作相关内容的报告和PPT |

|  |
| --- |
| 学号：2200330126 姓名：林镔壕 |
| 个人在团队中的贡献：完成课题“最佳分区”的部分代码。“可持续化线段树”的代码，以及制作相关内容的报告和PPT |

|  |
| --- |
| 学号：2200330129 姓名：刘康 |
| 个人在团队中的贡献：完成课题“最佳分区”的部分代码，以及制作相关内容的报告和PPT。 |

目 录

[1 课题介绍 1](#_Toc154271149)

[1.1 最佳分区 1](#_Toc154271150)

[1.2 生长树 1](#_Toc154271151)

[2 所用开发工具 2](#_Toc154271152)

[3 所使用的数据结构与算法 2](#_Toc154271153)

[3.1 最佳分区 2](#_Toc154271154)

[3.1.1 线段树 2](#_Toc154271155)

[3.1.2 排序 2](#_Toc154271156)

[3.1.3 动态规划 3](#_Toc154271157)

[3.2 生长树 3](#_Toc154271158)

[3.2.1 线段树 3](#_Toc154271159)

[3.2.2 邻接表 3](#_Toc154271160)

[3.2.3 深度优先搜索 3](#_Toc154271161)

[4 设计实现 5](#_Toc154271162)

[4.1 最佳分区 5](#_Toc154271163)

[4.1.1 init函数 5](#_Toc154271164)

[4.1.2 find函数 5](#_Toc154271165)

[4.1.3 pushup函数 6](#_Toc154271166)

[4.1.4 build函数 6](#_Toc154271167)

[4.1.5 query函数 7](#_Toc154271168)

[4.1.6 modify函数 7](#_Toc154271169)

[4.1.7 solve函数 8](#_Toc154271170)

[4.2 生长树 10](#_Toc154271171)

[4.1.1 build函数 10](#_Toc154271172)

[4.1.2 pushup函数 10](#_Toc154271173)

[4.1.3 pushdown函数 10](#_Toc154271174)

[4.1.4 modify函数 11](#_Toc154271175)

[4.1.5 query函数 11](#_Toc154271176)

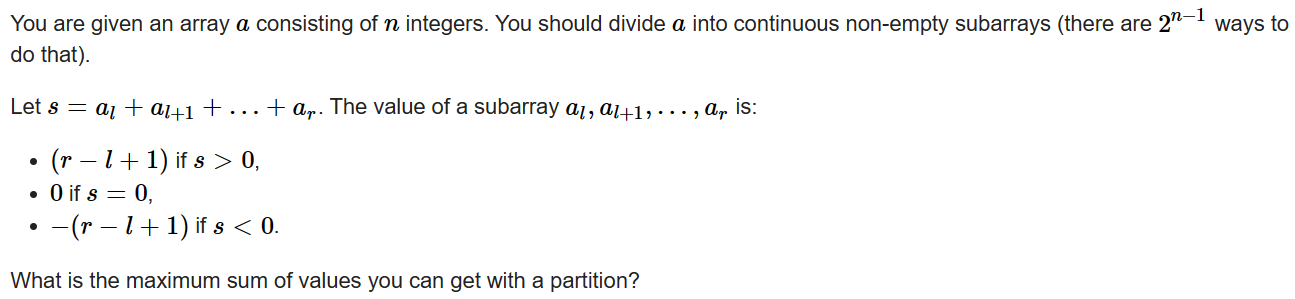
[4.1.6 dfs函数 12](#_Toc154271177)

[5 团队合作方式 13](#_Toc154271178)

[6 总结 13](#_Toc154271179)

1 课题介绍

1.1 最佳分区

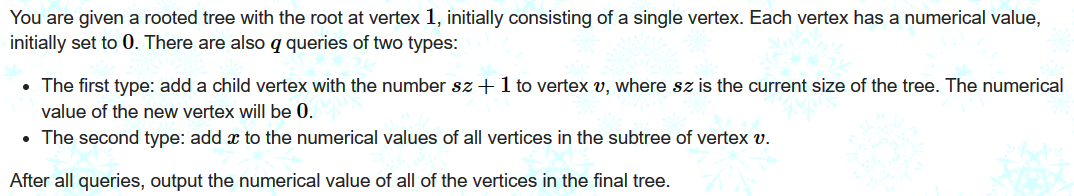


在此课题中，我们会得到一个数组，数组中包括n个整数。将该数组分成a个连续的非空区间（有2n-1个分类方法）。

令s = al + al+1 + . . .+ ar。

* 如果s > 0，该区间的值为：( r - l + 1 ) ；
* 如果s = 0，该区间的值为：0 ；
* 如果s < 0，该区间的值为：- ( r - l + 1 )

现需要求所有区间总和的最大值。

1.2 生长树

此课题要求我们模拟出一棵树，每个树节点都有一个编号(vertex)，和一个值(numerical value)；初始树节点编号为1，值为0。课题定义两种不同的节点修改方法作为输入：

1. 节点v增加一个孩子，编号为树节点数加1，值为0，输入方式：1 v
2. 让节点v及其所有子节点的值都加上x，输入方式：2 v x

在计算完所有树节点后，需要输出每个节点的值。

2 所用开发工具

Dev-C++，Visual Studio Code，git

3 所使用的数据结构与算法

3.1 最佳分区

该代码使用了线段树，排序和动态规划等算法。数据结构包括struct node表示线段树节点的结构体，vector<node> tr表示线段树，vector<ll> p表示前缀和数组，vector<ll> f表示经过排序去重后的前缀和数组。

3.1.1 线段树

线段树是一种非常有效的数据结构，主要用于处理区间查询问题。在该代码中，线段树被用来处理前缀和数组的区间查询操作。

在本问题中，希望找到两个不相交的子区间，使得第一个子区间的和减去第二个子区间的和的绝对值最大。为了实现这个目标，需要使用线段树来高效地查询区间的最大正值和最大负值。

线段树可以将整个输入数组划分为不同的区间，并且每个区间节点存储了该区间内的最大正值和最大负值。借助线段树的性质，可以快速地查询指定区间的最大正值和最大负值，并进行更新操作。

使用线段树的好处是可以将查询和更新操作的时间复杂度降低到O(log n)，其中n是数据的大小。相比于暴力遍历整个区间，线段树可以提供更高效的查询和更新操作，从而加快算法的执行速度。

3.1.2 排序

在该代码中，排序是为了构建经过去重的前缀和数组f。在解决问题时，我们需要在线段树中查询某个值在前缀和数组中的位置，通过使用排序可以将前缀和数组进行去重并按照升序排序，这样可以利用二分查找来快速找到指定值的下标。

具体来说，排序的目的是将前缀和数组中的元素按照升序排列，然后通过二分查找找到数组中某个值所在位置的下标。这样，可以在O(log m)的时间复杂度内找到值所在的位置，其中m是去重后前缀和数组的大小。

通过对前缀和数组进行排序和去重，我们可以在查询和更新操作中更快速地定位到指定数值在数组中的位置，从而提高算法的效率。

3.1.3 动态规划

在这段代码中，动态规划主要用于计算前缀和数组。动态规划是一种用于优化重复计算的技术，通过记录和利用之前的计算结果，以减少重复计算的次数。在这里，动态规划被用于计算前缀和数组p。

前缀和数组的计算可以用递推公式p[i] = p[i-1] + arr[i]进行，其中p[i]表示第i个位置的前缀和，arr[i]为输入数组的第i个元素。使用动态规划可以避免重复计算，而是利用之前计算得到的结果来计算当前位置的前缀和。

具体来说，当计算前缀和数组时，我们可以从数组的第一个元素开始，依次计算每个位置的前缀和，并将结果存储在前缀和数组中。通过动态规划，我们只需要计算每个位置的前缀和一次，并将结果存储起来，后续的计算就可以直接使用之前计算过的结果，而不需要重复计算。

这样，使用动态规划可以大大减少计算的时间复杂度，提高算法的效率。同时，前缀和数组的计算也为后续的问题解决提供了方便和便利的数据结构。

3.2 生长树

3.2.1 线段树

线段树说明如上。在本问题中，需要频繁地更新某一子树所有节点的值，并且所有值的增量相同。本问题还对时间做出了严格的规定。为了优化代码的时间复杂度，我们可以用线段树的节点来表示树的每个顶点，以及它的子树的数值之和。结合线段树的懒标记方法，可以减少遍历次数，从而减少算法的时间复杂度。

具体来说，每进行一次修改方法2，我们就可以用懒标记的方法来更新一个区间的所有顶点的数值，仅当在查询时，才将懒标记向下传递并更新子树节点的值，以减少所需时间。当需要查询某个节点的数值时，再遍历其父节点，下放懒标记，就能得到此节点的数值。

3.2.2 邻接表

邻接表是一种用于表示图的数据结构，它用一个数组或列表来存储每个顶点的邻接顶点。邻接表的优点是节省空间和时间，在图相对稀疏的情况下使用效果较好。在本题目中，邻接表被用于存储树的结构。

3.2.3 深度优先搜索

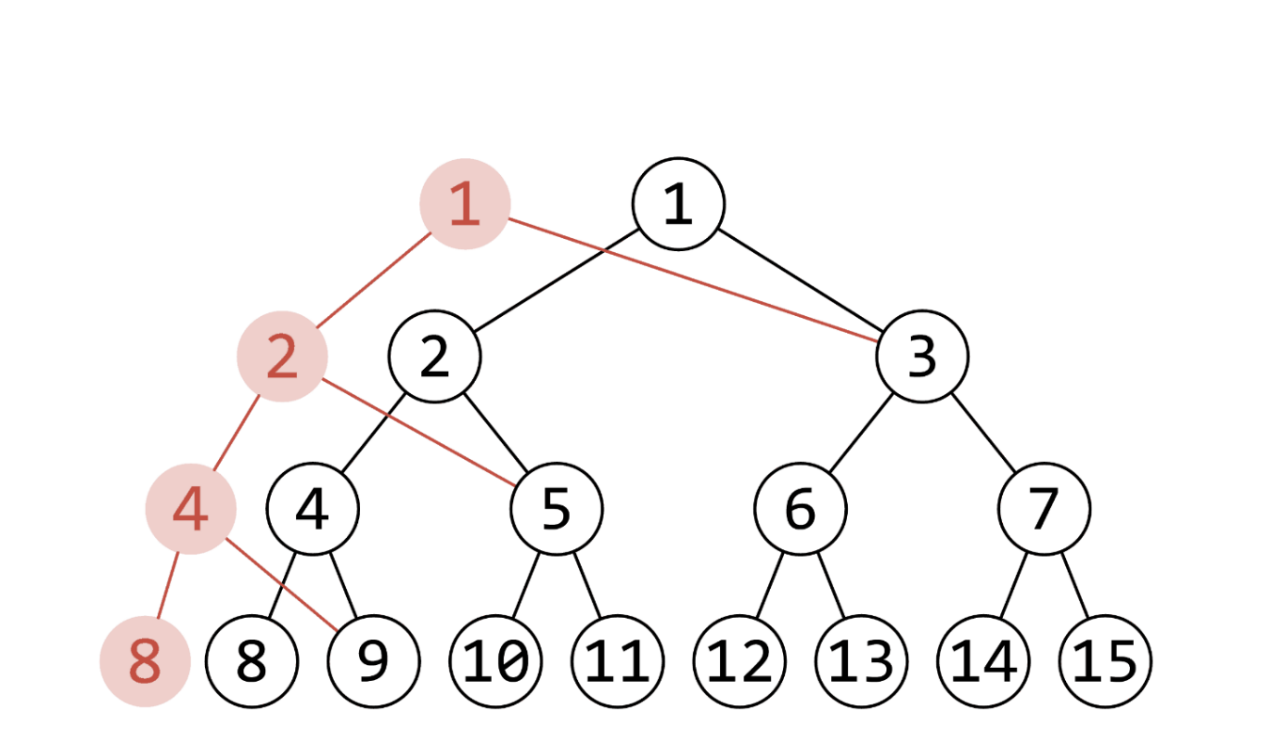
深度优先搜索（DFS）是一种用于遍历或搜索树或图的算法。这个算法会尽可能深地搜索树的分支。当节点v的所在边都已被探寻过，搜索将回溯到发现节点v的那条边的起始节点。这一过程一直进行到已发现从源节点可达的所有节点为止。

在本题目中，使用深度优先算法可以将图中的初始树节点以DFS序映射到线段树上，保证树上的任意两个节点之间的路径都是连续的区间，从而让线段树能够对原始树进行维护。

3.3 可持久化线段树

3.3.1 可持久化线段树简介

可持久化线段树是线段树的一种扩展，它在原有线段树的基础上引入了“可持久化”的概念。可持久化线段树允许在进行更新操作时，保留历史版本的数据结构状态，而不是直接修改原始数据结构。这种特性使得我们可以在不影响之前版本的基础上，对数据结构进行修改和查询。

3.3.2 可持久化线段树实现方式

如图所示，红色结点为单次进行修改操作变动过的结点，需要变动的结点会开辟新的空间，通过链式存储实现。没有经过修改的结点信息依然会被新的版本加以利用。且每次修改最多只会修改logn个结点，即约为树高，故修改操作为O(logn)时间复杂度与空间复杂度。

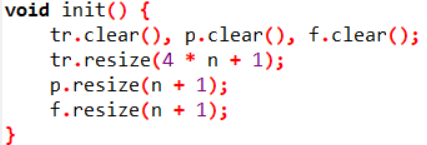
# 4 设计实现

4.1 最佳分区

本题目主要使用以下函数：

* init(): 初始化数据结构和容器；
* find(ll x): 在有序数组f中查找值x的位置；
* pushup(int u): 更新节点u的P和N值；
* build(int u,int l,int r): 构建线段树的节点u，覆盖区间为[l,r]；
* query(int u,int l,int r,int M): 查询线段树节点u中覆盖区间为[l,r]的最大值；modify(int u,int pos,int P,int N): 修改线段树节点u中位置为pos的最大值；
* solve(): 解题函数，根据输入数据执行求解过程。

4.1.1 init函数

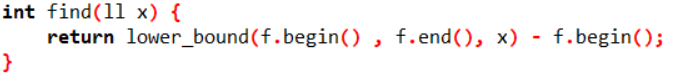


init函数的功能是初始化数据结构和容器。具体来说，它执行以下操作：

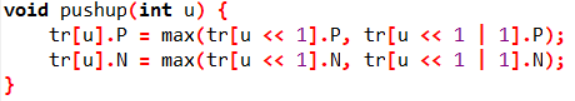
* 对全局向量tr，p和f 进行清空操作，即将其清空为一个空向量
* 重新调整全局向量tr的大小为4\*n+1，并将其初始化为全0
* 重新调整全局向量p的大小为n+1，并将其初始化为全0
* 重新调整全局向量f 的大小为n，并将其初始化为全0

通过以上的程序对全局向量tr，p和f 进行初始化。

4.1.2 find函数

该函数调用了标准库中的lower\_bound函数，用于在有序数组中查找第一个大于等于给定值的元素的位置。它返回的是指向该位置的迭代器。在 find函数中，使用lower\_bound(f.begin(), f.end(), x) 来查找 x 在数组 f 中的位置。然后通过计算lower\_bound返回的迭代器相对于数组首元素的偏移量，即可得到 x 在数组 f 中的下标位置。最后，将这个下标位置作为查找结果返回给调用函数。

4.1.3 pushup函数

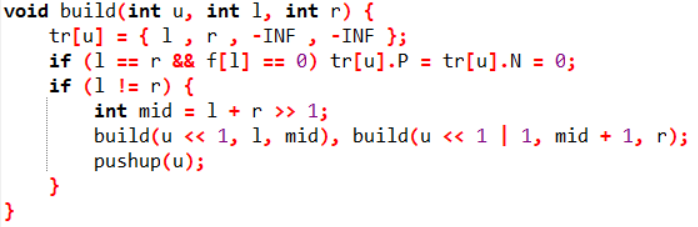
pushup 函数的功能是更新线段树中节点 u 的信息。在线段树中，每个节点都保存了一些与问题相关的信息。当子节点的信息发生改变时，父节点的信息也需要相应地进行更新。

具体而言，它会根据左右子节点的信息来更新节点 u 的信息。

在本代码实现中，节点 u 中保存了两个信息，即 P 和 N。P 表示节点 u 的正值最大值，N 表示节点 u 的负值最大值。更新节点 u 的信息意味着更新 P 和 N 的值。

函数在 pushup 中使用了左子节点和右子节点的信息，通过比较它们的值来更新节点 u 的信息。

4.1.4 build函数

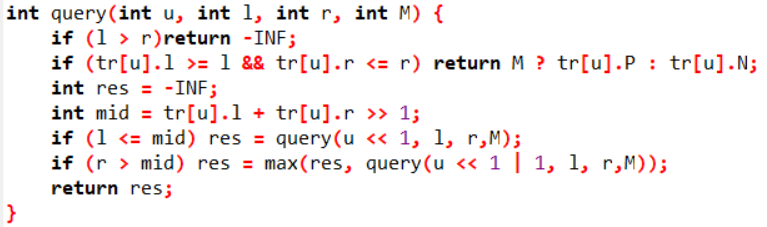
build函数的功能是在线段树中建立节点 u，并为该节点分配一个表示区间范围 [l, r] 的子树。

在函数的实现中，u 表示当前节点的编号，l 和 r 表示当前节点所对应的区间范围。函数首先判断 l 和 r 是否相等，若相等，则说明该节点是一个叶子节点，直接进行初始化操作。

如果 l 和 r 不相等，则说明该节点是一个非叶子节点，需要继续递归地构建其左子节点和右子节点。具体而言，函数会计算出当前节点的左孩子和右孩子节点的编号，然后调用 build 函数来构建左子树和右子树。

最后，函数会调用 pushup 函数来更新当前节点的信息，以确保节点的信息与左右子节点的信息一致。

4.1.5 query函数

query函数的功能是在线段树中查询区间 [l, r] 内满足特定条件 M 的信息。

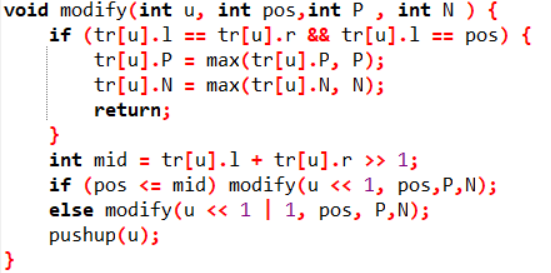
在线段树中进行区间查询时，一般会涉及到一些与具体问题相关的条件。在该函数中，参数 M 表示特定的条件。具体的条件可以根据实际应用而有所不同，比如查询区间内的最大值、最小值、总和等。

函数首先检查当前节点的区间是否完全包含在查询区间 [l, r] 内，如果是，则返回当前节点所保存的信息作为结果。这是因为当前节点的区间完全符合查询条件，无需继续向下进行查询。

如果当前节点的区间与查询区间存在交集，但不完全包含在查询区间内，则继续递归地查询其左子节点和右子节点。具体而言，会计算出当前节点的左孩子和右孩子节点的编号，并调用 query 函数来查询左子树和右子树中满足条件的信息。

最后，根据左子树和右子树的查询结果，结合特定的条件 M，计算出当前节点的信息，并返回给上一层的递归调用。

4.1.6 modify函数

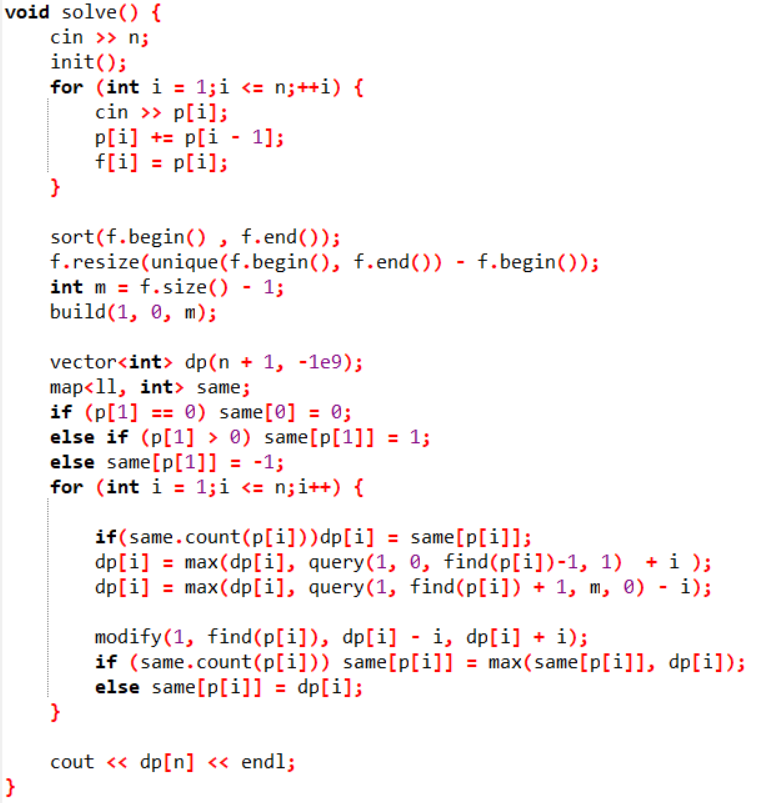
modify函数的功能是在线段树中修改位置 pos 处节点的信息。

在线段树中修改节点信息通常涉及到更新指定位置的节点，并相应地更新其父节点和祖先节点的信息。在该函数中，参数 pos 表示要修改的节点位置，P 和 N 表示要修改的节点的新值。

函数首先检查当前节点的区间是否包含位置 pos，如果包含，则将节点的信息更新为新值 P 和 N。这是因为当前节点即为要修改的节点，所以直接对其进行更新即可。

如果当前节点的区间不包含位置 pos，则根据位置 pos 的值判断是向左子节点还是向右子节点递归进行修改。具体而言，如果位置 pos 小于当前节点区间的中点，说明要修改的位置 pos 位于当前节点的左子树中，因此将递归调用 modify函数来修改左子树中的节点信息。反之，则在右子树。

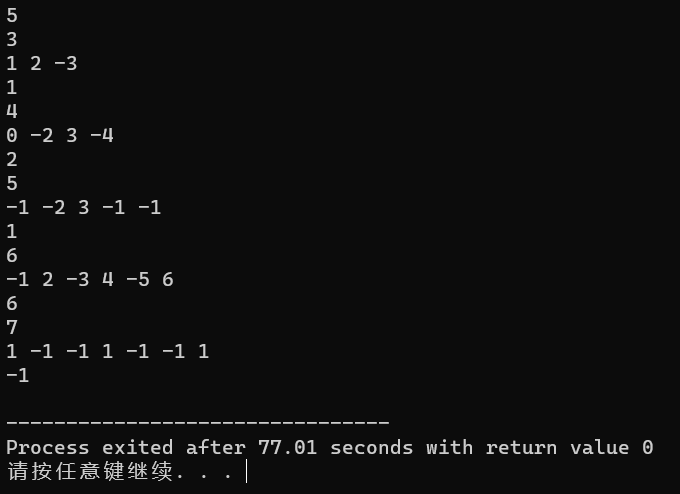
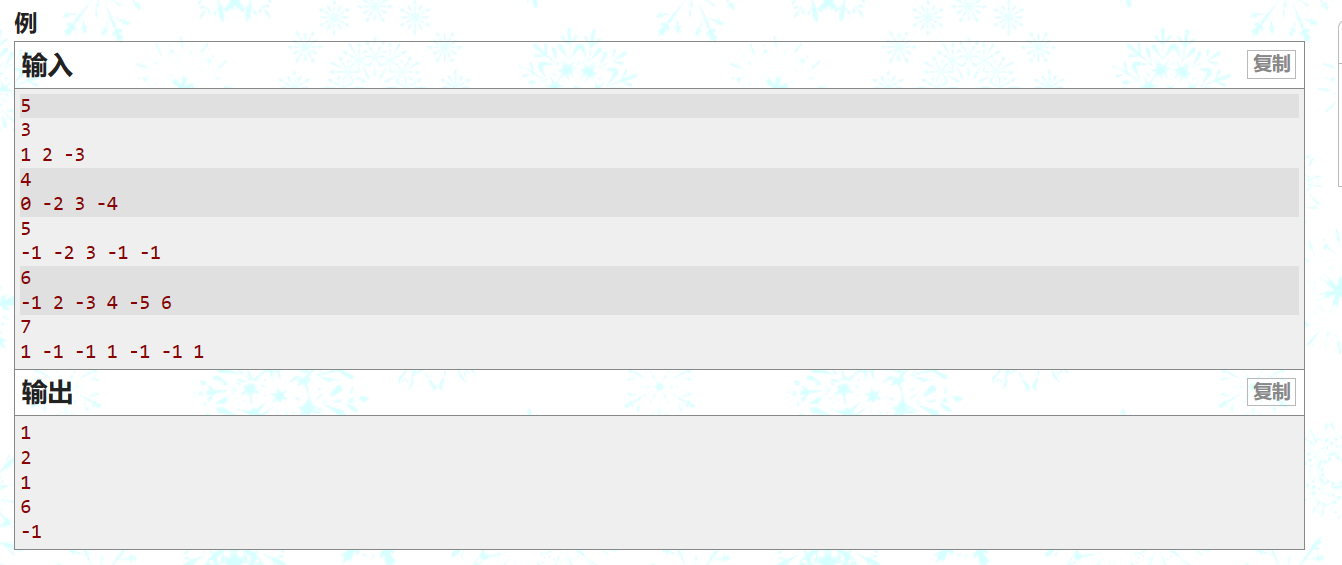
在递归调用完成后，需要调用 pushup 函数来更新当前节点及其祖先节点的信息。这是为了保证修改节点后，其父节点和祖先节点的信息仍然是准确的。

4.1.7 solve函数

solve函数是该代码的核心函数，其功能是解决具体的问题。下面对 solve 函数的具体步骤进行详细介绍：

1. 读入输入 n 的值，初始化全局变量。
2. 创建三个向量 tr、p 和 f，分别用于存储线段树的信息、前缀和和排序后的前缀和。
3. 通过循环，读取输入数据并计算前缀和。同时将前缀和存储在 p 向量中，将其作为 f 向量的元素。
4. 对 f 向量进行排序，并删除其中重复的元素。这样可以保证 f 向量是一个递增的有序数组。
5. 找到数组 f 的大小 m。调用 build(1, 0, m) 函数构建线段树。在线段树的每个节点上初始化左边界、右边界、最大值和最小值为负无穷（-INF）。
6. 创建长度为 n+1 的整型数组 dp，用于存储中间结果。初始化每个元素为负无穷（-1e9）。
7. 创建一个 map 类型的变量 same，用于存储相同值的最优解。
8. 判断前缀和数组中第一个元素的值，如果为0，将其存储在 same 中，并对应的值设置为0；如果大于0，将其存储在 same 中，并对应的值设置为1；如果小于0，将其存储在 same 中，并对应的值设置为-1。
9. 使用动态规划的思想，通过遍历前缀和数组 p，更新 dp 数组中的值。
10. 首先判断如果 same 中存在当前前缀和，将其对应的值赋给 dp。
11. 然后，通过调用 query(1, 0, find(p[i])-1, 1) 函数查询线段树中比当前前缀和小的最大值，并加上 i 的值更新 dp。
12. 再次，通过调用 query(1, find(p[i]) + 1, m, 0) 函数查询线段树中比当前前缀和大的最小值，并减去 i 的值更新 dp。
13. 最后，通过调用 modify(1, find(p[i]), dp[i] - i, dp[i] + i) 函数更新线段树中对应位置的值，并将当前位置的最优解存储在 same 中。
14. 输出 dp[n]，即最终的最优解。

4.1.8 程序运行结果

如图所示：代码运行的结果与课题所给出的运行样例一致，代码运行无错误。

4.2 生长树

本题目主要使用以下函数：

* build(int u, int l, int r): 对线段树进行初始化；
* pushup(int u): 更新当前节点u数值；
* pushdown(int u, int l, int r): 将节点u的懒标记下放；
* modify(int u, int l, int r, int ql, int qr, ll val): 线段树区间添加值val；
* query(int u, int l, int r, int ql, int qr): 查询线段树区间和；
* dfs(int u, int fa, vector<int>& rid, vector<int>& R,int &idx): 深度优先遍历,用于关联线段树和初始树；
* solve(): 解题函数，根据输入数据执行求解过程。

文本

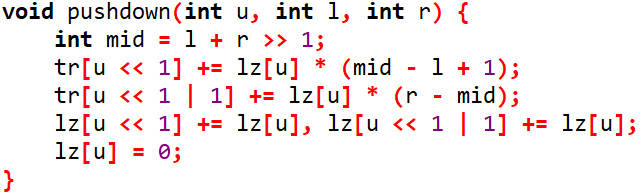
描述已自动生成4.1.1 build函数

build函数用来建立并初始化一个线段树，它的参数 u 表示当前节点的编号，l 和 r 表示当前节点所覆盖的区间的左右端点。这个函数会递归地建立左右子树，并初始化当前节点的数值和懒标记为0。

图表

描述已自动生成4.1.2 pushup函数

pushup函数用来更新一棵线段树的节点信息。具体来说，pushup函数就是由左右子节点的信息，更新当前节点u的信息。

4.1.3 pushdown函数

pushdown函数用来将线段树的节点的标记下传到其子节点。

在pushdown函数中，对线段树节点u，首先根据懒标记 lz[u] 更新其左右子节点的值，然后，将节点u的懒标记再下放给左右子节点，然后将节点u的懒标记清0，完成一次懒标记下放。

文本, 信件

描述已自动生成4.1.4 modify函数

modify函数用于寻找更新区间，为更新的线段树节点增加懒标记，并向上维护线段树。

在modify函数中，u指代线段树节点u，l和r为此线段树节点的管辖范围，ql、qr是所要修改的对应区间，val为这个区间中所有节点要增加的值。

当给定修改区间小于线段树修改区间时，modify函数仅对线段树区间进行修改，保留懒标记后离开函数。

当给定修改区间大于线段树修改区间时，首先先将线段树节点u的懒标记下放，然后再遍历使用左右子节点的区间来消化此区间修改。最后，将节点u的值更新，完成区间修改操作。

文本

描述已自动生成4.1.5 query函数

query函数用于查询区间之和。

首先，当给定线段树节点u的区间l、r要小于查询区间ql、qr时，直接返回线段树节点u的值；

如果给定线段树节点u的区间l、r要大于查询区间ql、qr时，就先将节点u的懒标记下放，然后向下继续遍历细分，最后累加得到区间ql、qr的值。

文本

描述已自动生成4.1.6 dfs函数

dfs函数用来将题目所给定树使用深度优先算法映射到线段树上。

在函数中，首先给当前节点u分配一个区间，即将rid[u]赋值为idx，表示u在先序遍历中的位置。

然后，遍历u的所有子节点i，如果i不是u的父节点fa，就递增idx，然后递归地调用dfs(i,u,rid,R,idx)，以给i及其子树分配区间。

最后，将R[rid[u]]赋值为idx，表示u的子树在先序遍历中的最后一个节点的位置。

函数完成后，每个线段树节点所指代的原树的区间就是[rid[v],R[v]]，完成映射。

文本

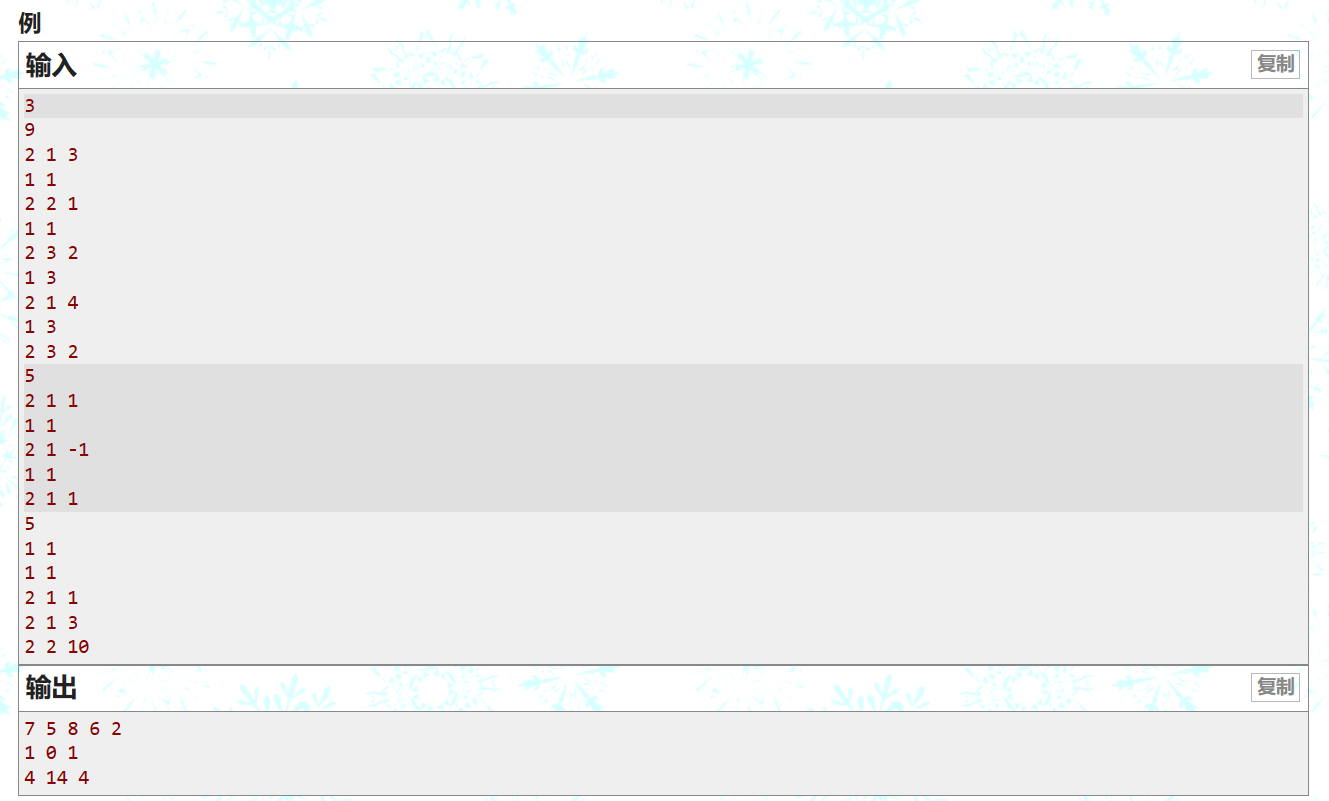
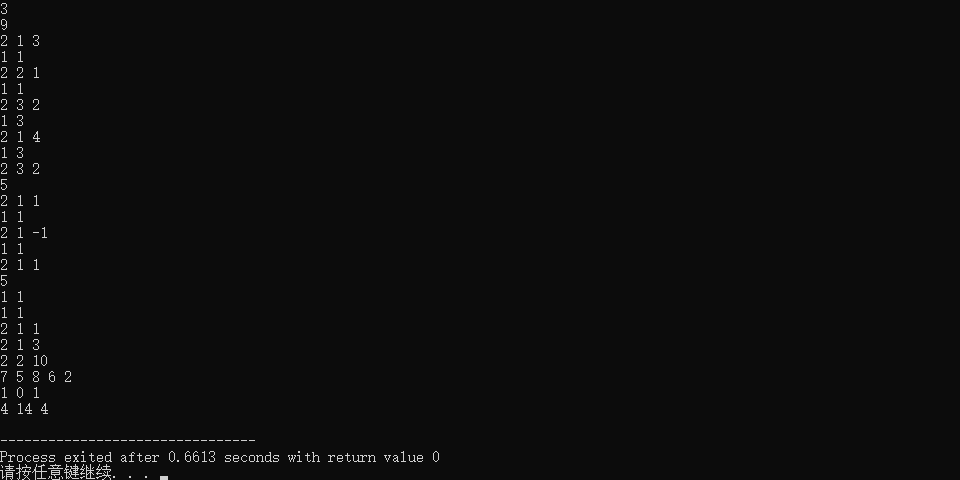
描述已自动生成4.1.7 solve函数

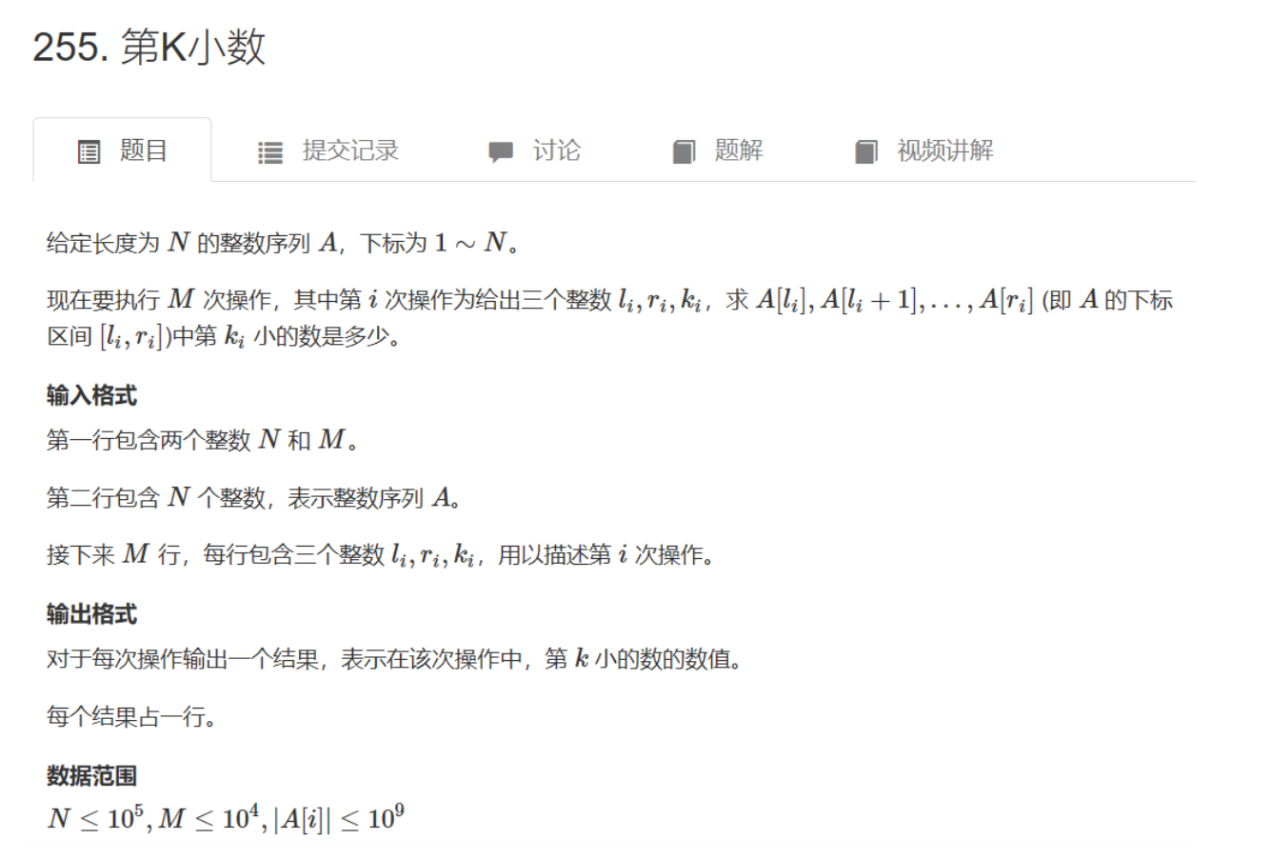
文本

描述已自动生成solve函数是该代码的核心函数，其功能是解决具体的问题。下面对 solve 函数的具体步骤进行详细介绍：

1. 读取一个整数q，表示操作的个数，然后清空g和op两个数组，用于存储树的结构和操作的信息。
2. 使用一个循环，从1到q，依次读取每个操作的类型和参数，并根据操作的类型，更新树的结构或记录操作的信息。如果操作的类型是1，表示添加一个子节点，就把新节点的编号加入到父节点的邻接表中，并把操作的类型和新节点的编号存入op数组。如果操作的类型是2，表示给子树加上一个值，就把操作的类型和参数存入op数组。
3. 调用build函数，建立一个大小为n的线段树，用于维护每个节点的数值和标记。初始时，每个节点的数值和标记都为0。
4. 调用dfs函数，从根节点开始，按照先序遍历的顺序，给每个节点分配一个区间，并记录在rid和R两个数组中。这样，每个节点v的区间就是[rid[v],R[v]]，用于线段树的查询和修改。
5. 使用一个循环，从1到q，依次处理每个操作。如果操作的类型是1，表示添加一个子节点，就把新节点的区间的数值和标记都置为0，即调用modify函数，给新节点的区间加上一个负的原来的数值。如果操作的类型是2，表示给子树加上一个值，就把子树的区间的数值和标记都增加相应的值，即调用modify函数，给子树的区间加上操作的参数。
6. 使用一个循环，从1到n，依次输出每个节点的数值，即调用query函数，查询每个节点的区间的数值之和并输出。

4.1.8 程序运行结果

如图所示：代码运行的结果与课题所给出的运行样例一致，代码运行无错误。

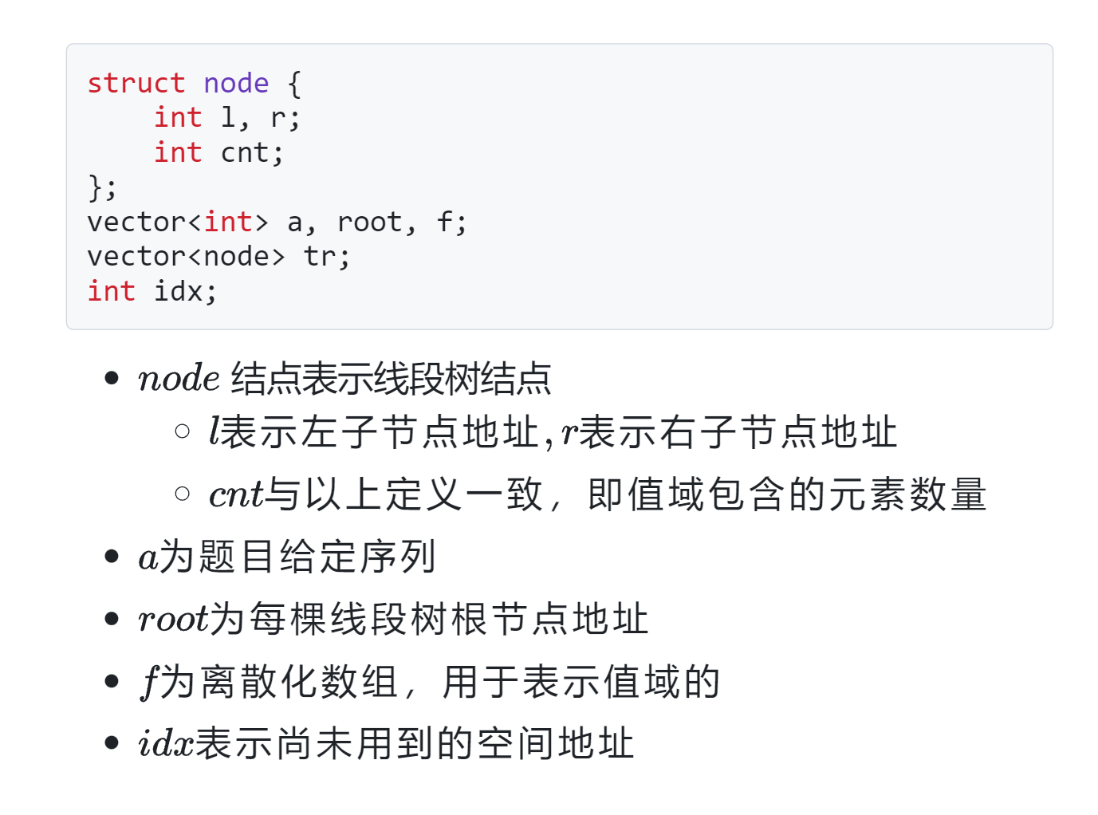
4.3 区间第k大值

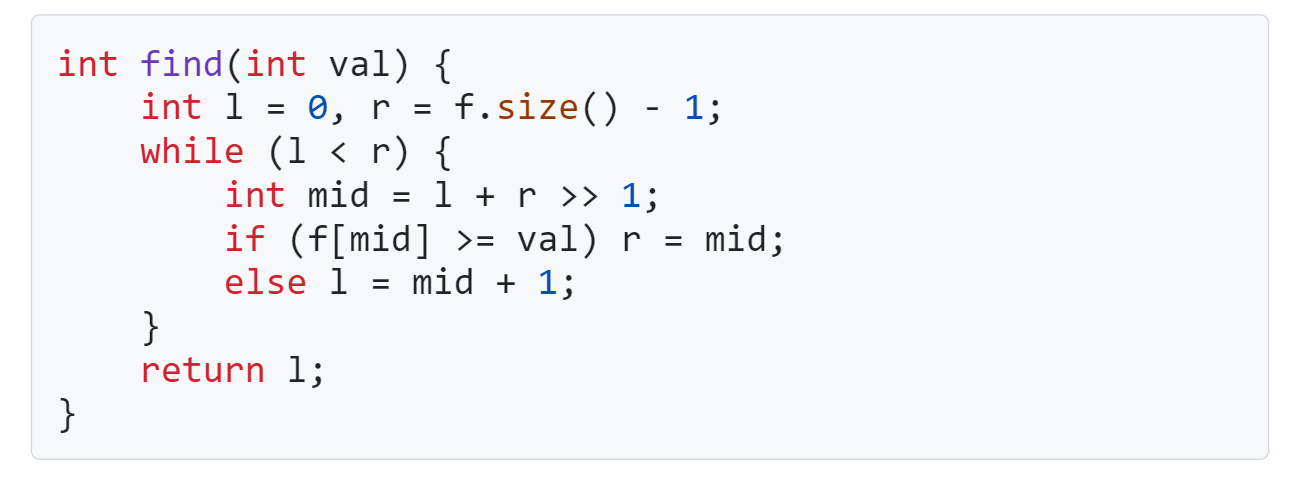
4.3.1 题目思路

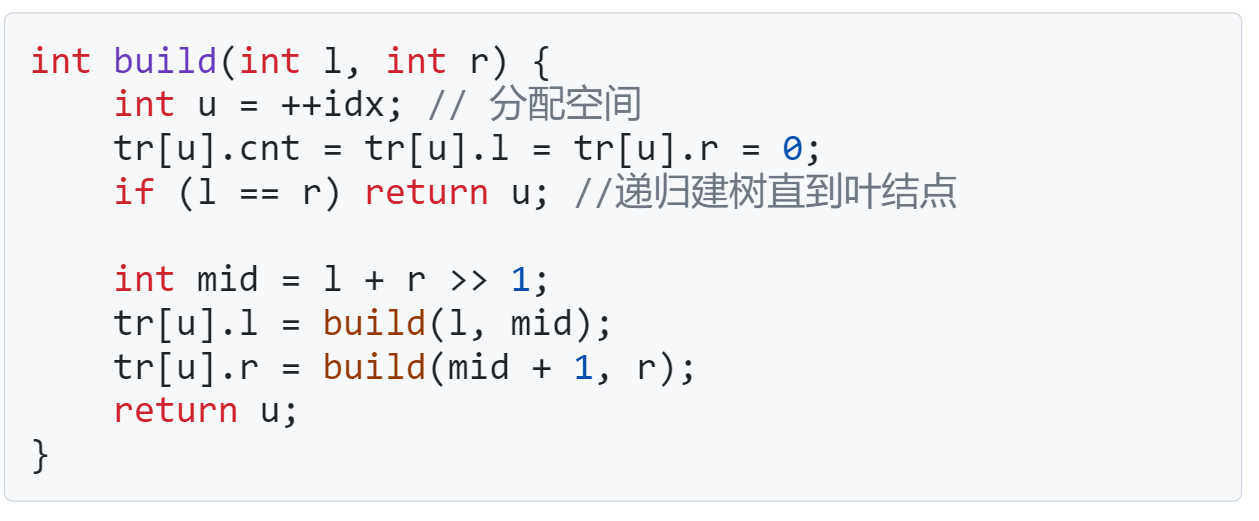
在经过离散化后的整数序列值域上建立一棵线段树，以表示某个具体值域内包含多少序列元素，借此便可通过二分的思想从线段树根节点开始搜索，若左子树包含元素小于查询的第K小数，递归查询左子树，反之查询右子树。并通过可持久化线段树建立n个线段树，每个版本从左到右依次插入整数序列元素，之后利用前缀和思想便可以求出特定区间包含的元素个数。

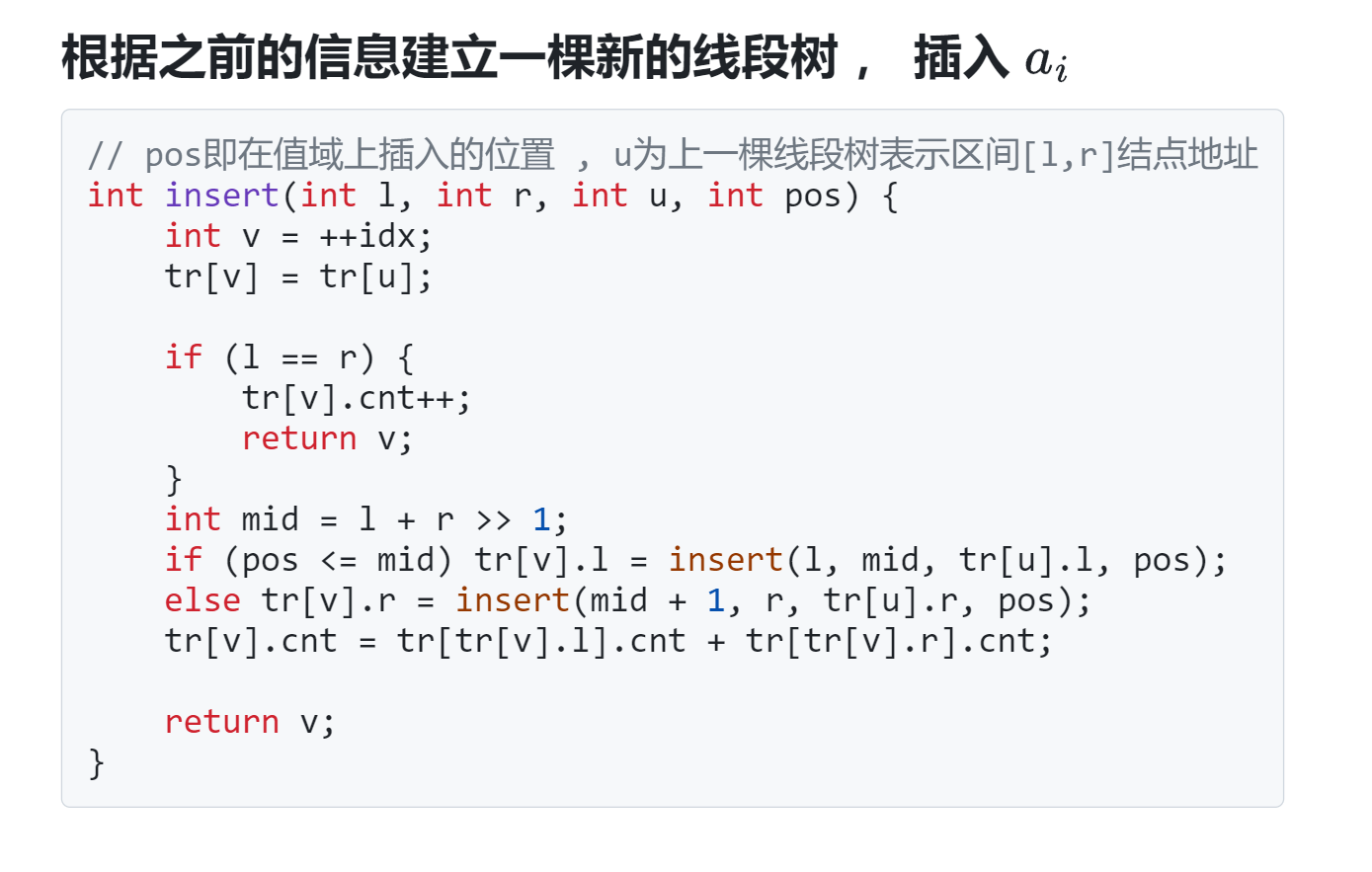
4.3.2 具体代码

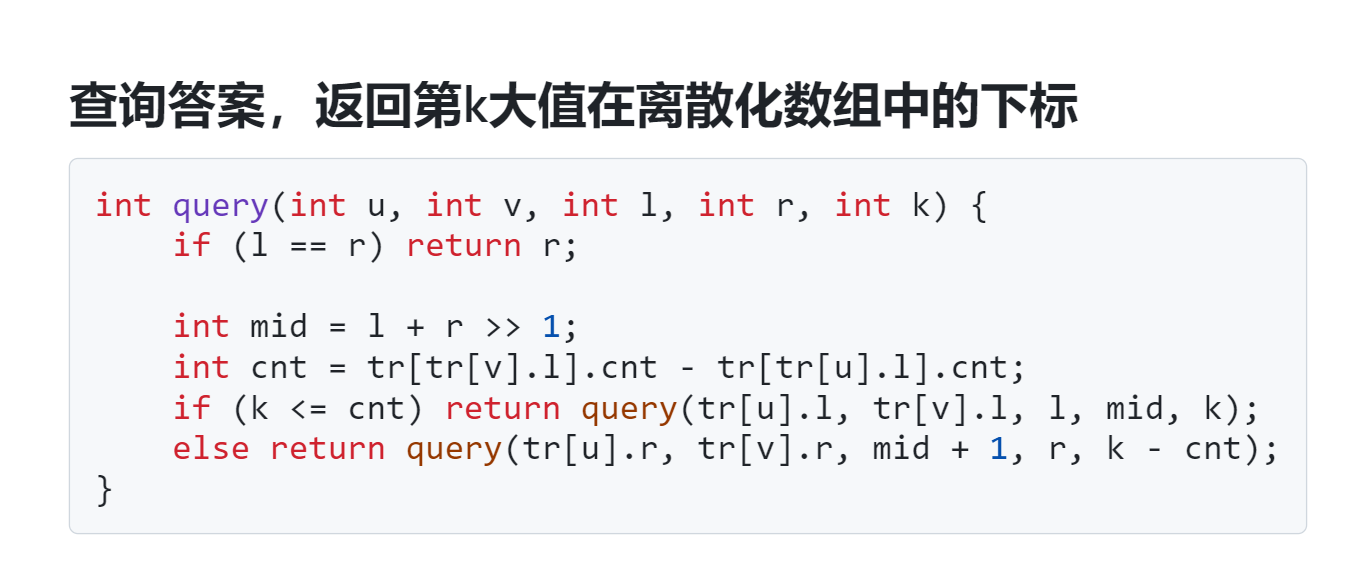
4.3.2.1 线段树结点与数组定义

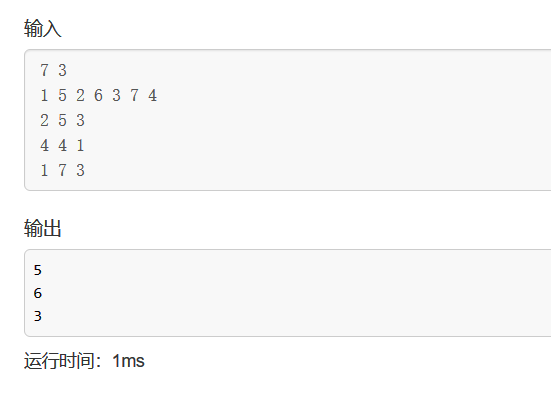


4.3.2.2 **二分搜索给定数值在离散化数组中的下标**

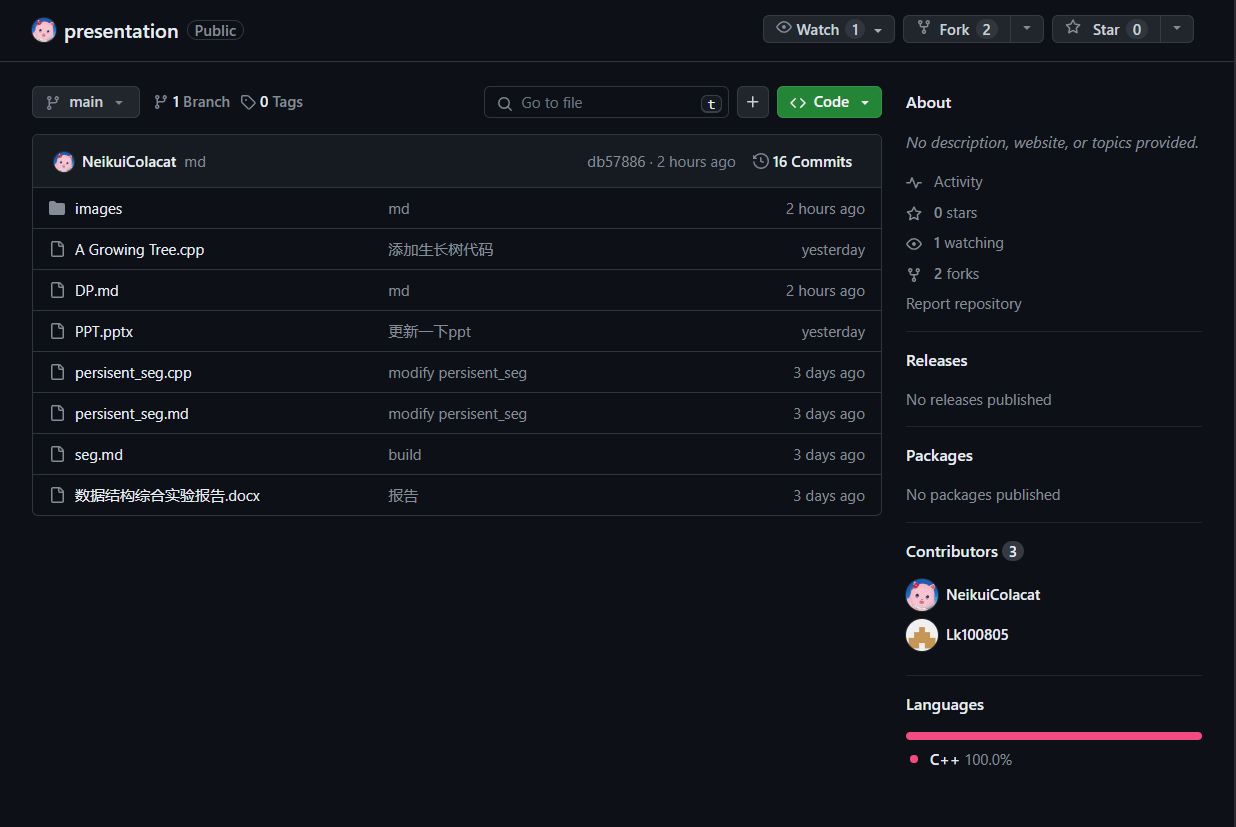
4.3.2.3 建立线段树

4.3.2.4 可持久化操作，更新版本

4.3.2.5 答案查询

4.3.3 样例输出结果

# 5 团队合作方式

在本次实验中，我们学习并使用了GitHub进行了代码托管以及文件存储。

GitHub是一个基于git的在线代码托管平台，它可以让开发者在同一个项目中共享、修改和合并代码。GitHub提供的一些工具和功能，也让我们的团队协作更加高效和便捷。

在我们进行代码编写时，使用git进行版本管理和分支化工作，让我们的代码可以更快更好地实现高效部署和统一化，并且在误操作发生时，还可以进行版本回退来快速地重新解决问题；当遇到两人代码冲突时，我们也可以借助GitHub平台很便捷地进行冲突比对和合并，十分方便。

除了使用GitHub平台外，我们在线上线下也积极主动多次交流，提出目前遇到的问题和挑战，给出个人的经验和总结，和善积极地进行团队沟通也是我们完成本实验的关键。

# 6 总结

6.1 最佳分区

在这个实验中，我们使用线段树来存储前缀和的信息，并通过动态规划的方法来解决最优解。动态规划的核心思想是将问题划分为较小的子问题，并通过子问题的解来计算更大规模问题的最优解。而线段树则为我们提供了一种高效地存储和查询区间信息的数据结构。

我们首先计算数组的前缀和，这样将问题转化为了区间和的计算问题。然后，我们使用线段树来存储前缀和数组的信息。构建线段树的过程中，我们将每个节点表示为一个区间范围，并存储该区间范围内的和。通过构建线段树，我们可以在 O(logN) 的时间内查询和修改任意区间范围内的和。

接下来，我们使用动态规划的方法来计算最优解。我们定义一个状态数组 dp，其中 dp[i] 表示前 i 个元素的最优解。通过遍历前缀和数组，我们更新状态数组 dp 的值。在更新的过程中，我们利用线段树查询区间和的操作来计算给定区间范围内的和，然后使用动态规划的思想更新最优解。

最后，我们输出最终的最优解，即 dp 数组的最后一个元素。通过这个过程，我们可以高效地求解给定数组的最优解。

总结起来，线段树和动态规划是一对强大的组合，尤其适用于解决范围查询和最优解问题。线段树提供了高效的区间查询和更新操作，而动态规划通过分解问题和状态定义来计算最优解。在这个实验中，我们充分利用了线段树和动态规划的特性，成功解决了特定的问题

6.2 生长树

在这次实验里，我们主要使用了线段树来解决了两个问题，一个是最佳分区问题，第二个是生长树问题。这两个问题在修改时都会涉及到大量的子树修改，使用线段树的特性可以很好地优化在解决这两个问题时的时间复杂度。除了线段树之外，我们还使用了其他的数据结构和算法，例如动态规划、邻接表、深度优先映射等，来帮助我们实现所需目标。

在本次实验里，我们不仅收获了新的数据结构和算法的知识，提升了自我的实践能力，还学会了使用GitHub进行团队项目的管理，这对我们而言是巨大的收获。

6.3 第k小值

对于该题目使用线段树再加上可持久化数据结构的思想，每次进行线段树的更新的时候利用之前的信息来维护，使得修改操作不需要额外复制一棵新的线段树，降低了空间复杂度。