

综合实验报告

课题名称： 线段树

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 14 日

|  |
| --- |
| 学号： 姓名： |
| 个人在团队中的贡献： |

|  |
| --- |
| 学号： 姓名： |
| 个人在团队中的贡献： |

|  |
| --- |
| 学号： 姓名： |
| 个人在团队中的贡献： |

目 录

（三号、黑体、居中、目录两字空四格、与正文空一行）

1（空两格）☆☆☆☆，☆☆（四号黑体）**……………………………**1

1.1（空一格）☆☆☆，☆☆☆（小四号黑体）**………………………………………**

目 录

[1 课题介绍 1](#_Toc13715)

[1.1 最佳分区 1](#_Toc3070)

[2 所用开发工具 1](#_Toc28284)

[3 所使用的数据结构与算法 1](#_Toc22963)

[3.1 最佳分区 1](#_Toc21552)

[3.1.1 线段树 1](#_Toc5058)

[3.1.2 排序 2](#_Toc23965)

[3.1.3 动态规划 2](#_Toc25924)

[4 设计实现 3](#_Toc25051)

[4.1 最佳分区 3](#_Toc10211)

[4.1.1 init()函数 3](#_Toc32526)

[4.1.2 find(ll x)函数 3](#_Toc10168)

[4.1.3 pushup(int u)函数 4](#_Toc25963)

[4.1.4 build(int u,int l,int r)函数 4](#_Toc32435)

[4.1.5 query(int u,int l,int r,int M)函数 5](#_Toc3942)

[4.1.6 modify(int u,int pos,int P,int N)函数 5](#_Toc3223)

[4.1.7 solve()函数 6](#_Toc30806)

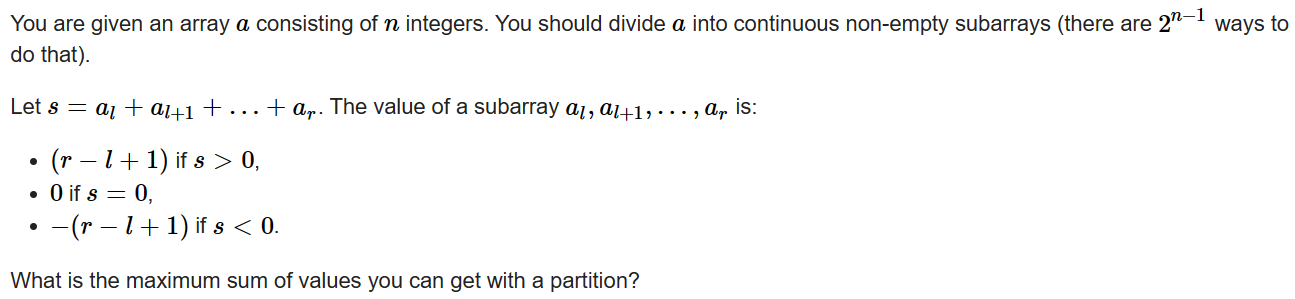
[5 团队合作方式 7](#_Toc30616)

[6 总结 8](#_Toc11524)

[参考文献 8](#_Toc17174)

1 课题介绍

1.1 最佳分区



在这个课题中，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。

2 所用开发工具

Dev-C++，Visual Studio Code

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。

## 4 设计实现

4.1 最佳分区

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。

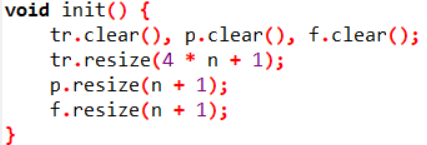
pushup(int u):更新节点u的P和N值。

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

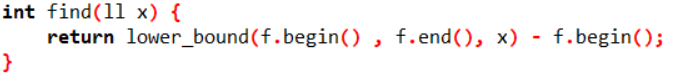


init()函数的功能是初始化数据结构和容器。具体来说，它执行以下操作：

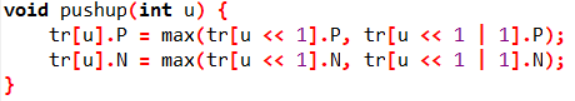
* 对全局向量tr，p和f 进行清空操作，即将其清空为一个空向量
* 重新调整全局向量tr的大小为4\*n+1，并将其初始化为全0
* 重新调整全局向量p的大小为n+1，并将其初始化为全0
* 重新调整全局向量f 的大小为n，并将其初始化为全0

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(ll x)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(int u)函数

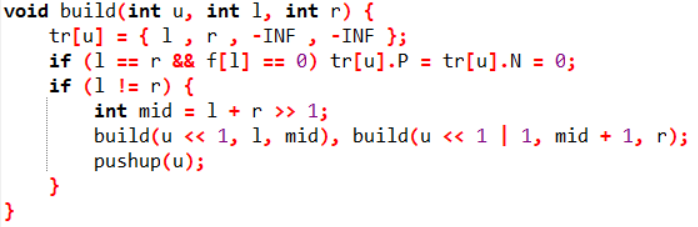
pushup(int u) 函数的功能是更新线段树中节点 u 的信息。在线段树中，每个节点都保存了一些与问题相关的信息。当子节点的信息发生改变时，父节点的信息也需要相应地进行更新。

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(u) 中使用了左子节点 u << 1 和右子节点 u << 1 | 1 的信息，通过比较它们的值来更新节点 u 的信息。

4.1.4 build(int u,int l,int r)函数

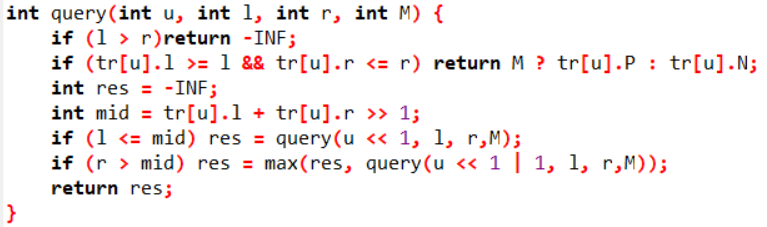
build(int u, int l, int r) 函数的功能是在线段树中建立节点 u，并为该节点分配一个表示区间范围 [l, r] 的子树。

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(int u,int l,int r,int M)函数

query(int u, int l, int r, int M) 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。

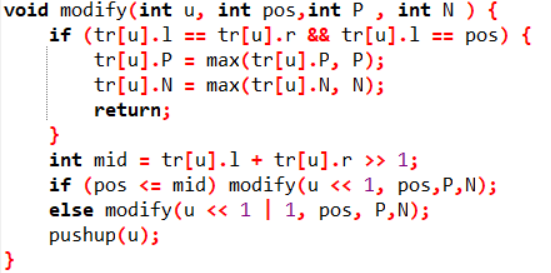
在线段树中进行区间查询时，一般会涉及到一些与具体问题相关的条件。在该函数中，参数 M 表示特定的条件。具体的条件可以根据实际应用而有所不同，比如查询区间内的最大值、最小值、总和等。

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(int u,int pos,int P,int N)函数

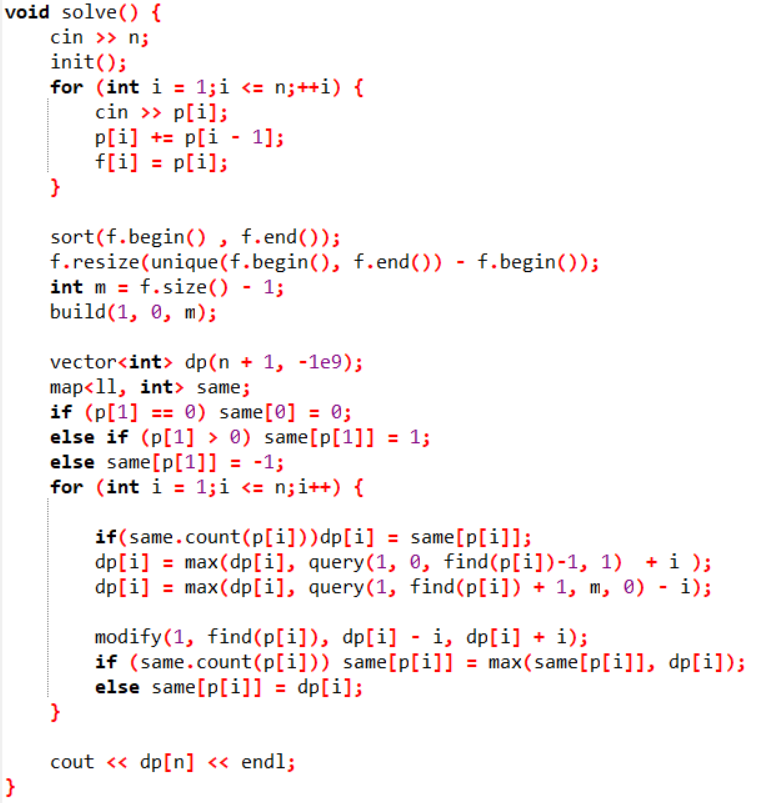
modify(int u, int pos, int P, int N) 函数的功能是在线段树中修改位置 pos 处节点的信息。

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。

## 5 团队合作方式

团队分工合作，要求每位团队成员都要进行实际代码开发，可以由组长进行模块划分以及工作分配。可以考虑使用gitee进行代码托管和研发协作。

## 6 总结

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