DSAP110-2 期末報告

B10705024 楊中

June 2022

1 簡介線段樹

- 甚麼是線段樹?

  線段樹是一種以區間為單位儲存資料的結構，其最大特色為可以快速查詢、修改結構內任一區間所儲存的資料，並且線段樹是 height-balanced 的樹。

- 線段樹能做甚麼?

  線段樹可用於區間求和、區間求最大值、區間求最小值、多個矩形互相覆蓋的面積等等的問題。

- 線段樹的優點?

  線段樹最大的優點在於當我們有修改區間的需求，且想查詢的範圍很多而不固定時，我們可以對每一筆資料用 $O(\log n)$ 的時間複雜度來查詢、修改，如果用暴力解慢慢遍歷陣列元素，則每筆資料須 $O(n)$。

2 範例

2.1 區間最大值

- 問題：給定一個數列 $T_1, T_2, T_3, ..., T_n$，求 T_a T_b(含邊界) 的最大值？來源
- 例：給定陣列 [3, 2, 4, 5, 6, 8, 1, 2, 9, 7]，求 $[2, 7], [1, 5], [3, 9], ...$ 的最大值
  - 暴力解

    直接根據每筆欲查詢的區間去遍歷原陣列，最壞每次查詢都需 $O(\log n)$ 的時間複雜度。

    Pseudo Code、Source Code

```
for(int i = 0; i < testcase; i++){
        int left, right = 0;
        cin >> left >> right;
        int max = arr[left-1];
        for(int j = left; j <= right-1; j++){
                if(max < arr[j])
                        max = arr[j];
        }
        cout << max << '\n';
}
```

圖 1: 暴力解虛擬碼

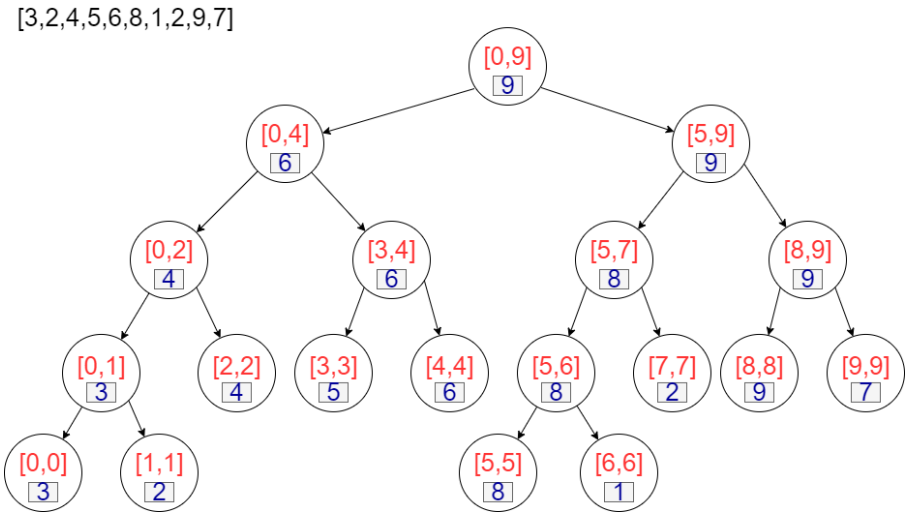

圖 2: 線段樹求最大值示意圖

– 線段樹解

如上圖所示，假如要查詢區間 [2, 9]，首先會從根部開始查詢。[2, 9] 範圍不等於當前節點的 [0, 9]，因此會將欲查詢區間切割成 [2, 4] 和 [5, 9] 分別往左子樹和右子樹查詢。再來第二步，在右子樹的地方，因為欲查詢的 [5, 9] 等於當前節點的範圍，所以直接回傳當前節點所儲存的最大值即可；而在左子樹的地方，則因範圍與當前節點不同，所以會重複在第一步時發生的事情，分別向左右子樹查詢。最後取所有回傳值中最大的那個，即是區間 [2, 9] 的最大值。如此一來即可得到時間複雜度為 $O(\log n)$ 的解。

2.2 矩形覆蓋面積

- 問題：給定 n 個在平面上的矩形，求它們在平面上的覆蓋面積？

– 先備知識

1. 離散化－將原本的數值按大小順序重新編碼
2. 掃描線－想像有一條無限長的線從平面上掃過去，將圖形重新分割為一條條的矩形

– 解法

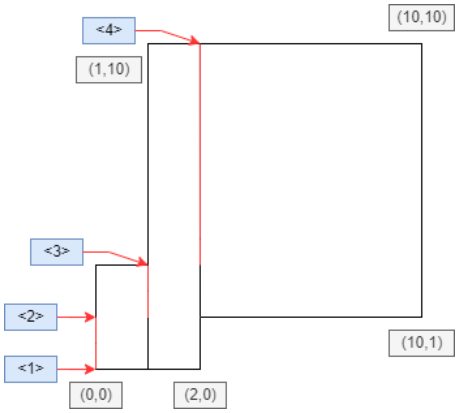

圖 3: 矩形覆蓋面積示意圖

如上圖所示，我們先以離散化的手法將 y 值重新編號，再利用掃描線的觀念將這兩塊重疊的矩形看作三條長方形。接著，我們依照 x 座標的大小順序讀入一條條的直線，並將矩形的左

邊稱為入邊，右邊稱為出邊。這時，我們會透過出邊和入邊的判斷來控制當前這塊矩形的高，入邊 $+1$ 而出邊 -1。透過這樣的方式，我們就能透過線段樹及時更新當前矩形的高，並不斷算出當前矩形的面積，最後只要把這些面積一塊塊加總便是答案了。

– Source Code

3 比較與相似資料結構的異同

3.1 二元搜尋樹

- 二元搜尋樹維護資料間的大小關係，具有左子樹的資料必小於父節點而右子樹的資料必大於父節點的性質；而線段樹則是依照區間的分割維護儲存的資料，左子樹儲存左半區間的資料而右子樹儲存右半區間的資料，父節點則儲存整段區間的資料。
- 空間複雜度上兩者同為 $O(n)$；平均時間複雜度上，二元搜尋樹的查詢、插入、刪除皆為 $O(\log n)$，但在最壞時間複雜度上則均為 $O(n)$；線段樹則在單點查詢、區間查詢、單點修改、區間修改上，無論平均或最壞，時間複雜度均為 $O(\log n)$

3.2 稀疏表

- 稀疏表藉由預處理的方式，對於一組靜態資料進行快速查詢；而線段樹則支援快速修改的功能。
- 稀疏表在預處理時，需花費 $O(n \log n)$ 的時間、空間複雜度，但在查詢時則有 $O(1)$ 的超快速查詢；線段樹在建樹時，需花費相對較快的 $O(n)$ 時間複雜度和較小的空間複雜度 $O(n)$，在查詢時的表現則較為遜色，需花費 $O(\log n)$ 的時間複雜度。

3.3 二元索引樹

- 二元索引樹用於快速計算字首和、區間和；而線段樹同樣也包含此功能，但是線段樹可以處理的問題更加廣泛，比如說計算區間最大值等。
- 二元索引樹在建樹、查詢、修改上的時間複雜度都和線段樹一樣為 $O(\log n)$；而建樹所耗的空間複雜度也同樣為 $O(n)$，唯常數上有些不同，二元索引樹為 n，而線段樹則為 $4n$。

4 制定程式介面與實作

4.1 介面

- void segmentTree(inputs)：建構子，負責建立整顆樹。
- int query(range)：查詢函式，可查詢感興趣的區間。
- void add(value, range)：修改函式，可在指定的單點、區間同時加上某個值
- void assignment(value, range)：修改函式，可將指定的單點、區間都變為某個值

4.2 實作

- void segmentTree(inputs)

  如下圖所示，建構子只需要透過向左子樹和右子樹遞迴，並在最後將子節點的結果合併回傳給父節點，即可成功建樹。(建構子中僅呼叫此處的 buildTree 而已)

```cpp
void SegmentTree::buildTree(vector<int>& v, int vertex, int l, int r)
{
    if(l == r){
        maxVal[vertex] = v[r];
        minVal[vertex] = v[r];
        sumVal[vertex] = v[r];
        return;
    }
    int middle = (l + r) / 2;
    buildTree(v, vertex * 2 + 1, l, middle);
    buildTree(v, vertex * 2 + 2, middle + 1, r);
    //將子節點的結果合併後回傳給父節點
    push_up(vertex);
}
```

圖 4: 建構子函式

- int query(range)

  如下圖所示，查詢函式只需要透過遞迴的方式查找。如果欲查詢範圍完全在左邊，那就往左子樹遞迴；如果欲查詢範圍完全在右邊，那就往右子樹遞迴；如果橫跨兩邊，那就切割欲查詢範圍後再分別往左右遞迴。最後向上合併回傳結果即可。

```cpp
int SegmentTree::maxi(int vertex, int ql, int qr, int l, int r)
{
    // if(l != r)//當前區間為一個點，不可再往下找最大值(因為該點為leaf)
    //     push_up(vertex);

    if(ql == l && qr == r){//當前節點範圍等同欲查詢區間，直接回傳結果
        return maxVal[vertex];
    }
    push_down(vertex, l, r);//將lazy tag往下傳遞

    int middle = (l + r) / 2;
    if(qr <= middle){
        return maxi(vertex * 2 + 1, ql, qr, l, middle);/*如果欲查詢的區間範圍完全在前半段*/
    }else if(ql > middle){
        return maxi(vertex * 2 + 2, ql, qr, middle + 1, r);/*如果欲查詢的區間範圍完全在後半段*/
    }else{
        /*如果欲查詢的範圍橫跨左右半段*/
        return std::max(maxi(vertex * 2 + 1, ql, middle, l, middle), maxi(vertex * 2 + 2, middle + 1, qr, middle + 1, r));
    }
}
```

圖 5: 查詢函式

- void add(value, range)、void assignment(value, range)

  由於加值和賦值修改的性質相近，因此以下僅以加值函式為例。在處理修改函式時，要特別注意，如果我們是做單點修改，那應該有個時間複雜度為 $O(\log n)$ 的簡單遞迴方法。但在處理區間修改時，如果用對 n 個點做單點修改的方式來達到區間修改的效果的話，時間複雜度將會來到恐怖的 $O(n \log n)$。所以在此我們要引進一個新的方法-懶人標記 (lazy tags)。

  懶人標記的核心思想在於，當修改函式被呼叫了，我們只遞迴到相應要被修改的區間，並在此處打上象徵「待修改」的標記後便結束函式。這個所謂的「修改函式」概念上並不會去改變區間的值，因為我們只需要在使用者真的去呼叫查詢函式時，再修改區間值並回傳正確結果即可。因此懶人標記的運作方式，即是等到其他函式運行的時候，再順便沿著遞迴的路徑，沿途修改區間值，以此達到區間修改的效果。換句話說，懶人標記的精神，就是把自己該做的事情丟出來，請其他人在他們工作時順便幫自己把事做完，從而大幅減少自己所需花費的時間。

```cpp
void SegmentTree::updateAdd(int vertex, int ql, int qr, int l, int r, int addVal)
{
    if(ql == l && qr == r){//當前區間為欲更新區間，直接更新後結束
        lazyAdd[vertex] += addVal;
        maxVal[vertex] += addVal;
        minVal[vertex] += addVal;
        sumVal[vertex] += addVal * (r - l + 1);
        return ;
    }
    push_down(vertex, l, r);//向下傳遞lazy tag
    int middle = (l + r) / 2;
    if(qr <= middle){/*如果欲查詢的區間範圍完全在前半段*/
        updateAdd(vertex * 2 + 1, ql, qr, l, middle, addVal);
    }else if(ql > middle){/*如果欲查詢的區間範圍完全在後半段*/
        updateAdd(vertex * 2 + 2, ql, qr, middle + 1, r, addVal);
    }else{
        /*如果欲查詢的範圍橫跨左右半段*/
        updateAdd(vertex * 2 + 1, ql, middle, l, middle, addVal);
        updateAdd(vertex * 2 + 2, middle + 1, qr, middle + 1, r, addVal);
    }
    //向上合併子節點的結果
    push_up(vertex);
}
```

圖 6: 加值函式

如上圖所示，我們只需簡單的遞迴相應要修改的區間，並在該處打上懶人標記即可。圖中的 push_down 函式，效果即為向下傳遞一層懶人標記。此外要注意的是，使用懶人標記後一定要記得將結果合併回傳給父節點 (push_up 函式)。並且加值和賦值的優先順序須備特別注意，這個順序可取決於具體需要而定。

5 複雜度分析

- 建構子，空間複雜度-$O(n)$、時間複雜度-$O(n)$

  假設第一層有一個節點，第二層兩個、第三層四個...。以此類推，得到總節點數為 $1+2+4+8+...+2^{\lceil \log n \rceil} < 2^{\lceil \log n \rceil + 1} < 4n$，故空間複雜度為 $O(n)$。時間複雜度為 $O(n)$ 的原因相同。

- 查詢函式，輔助空間複雜度-$O(\log n)$、時間複雜度-$O(\log n)$

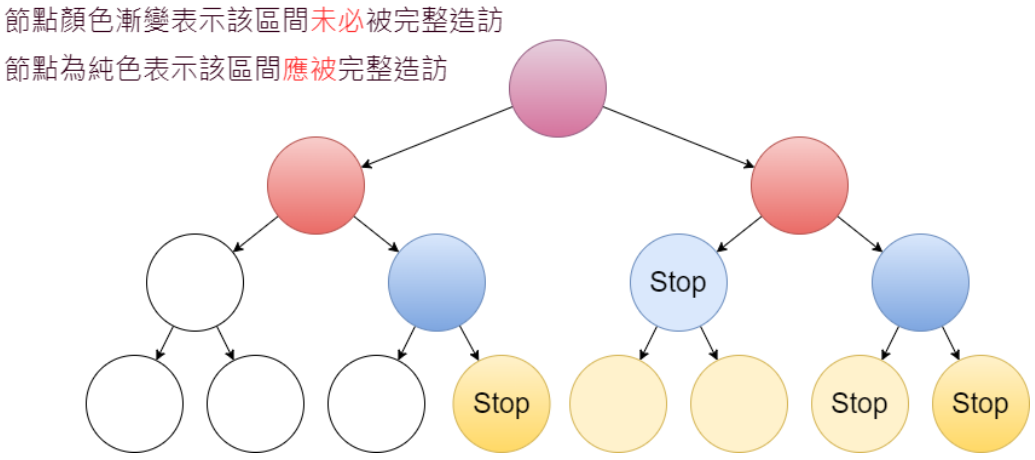

圖 7: 造訪過程示意圖

仔細觀察查詢過程，會發現最多有 $4\log n$ 個點被造訪到。如上圖所示，假如要在同一層同時造訪到五個點的話，會發現一定有其中兩個點的範圍可以被他們的父節點所包含，因此不可能在同一層同時造訪到五個點。

- 修改函式，輔助空間複雜度-$O(\log n)$、時間複雜度-$O(n)$

  因為具體實現方式為遞迴到相應欲修改區間，因此複雜度同查詢函式複雜度的分析。

6 更多應用

- 要求：可分治性！
- 舉例：最大公因數、最小公倍數、牆面染色問題等等
  - 線段樹和二元搜尋法

    如要查詢 [1, N] 之間第一個不小於 K 的數，在二元搜尋法中我們會利用中間那個元素的大小，來判斷要往左找或往右找；而在線段樹中，我們可以利用區間的最大值來決定要往左或往右，如果當前節點的左子樹的區間的最大值小於 K，那就往右找。反之，則往左。如果兩邊的最大值都不小於 K，那就往左找，因為我們要「第一個」。
  - 高維線段樹-樹包樹

    可藉此維護二維平面，乃至於更高維的空間。查詢複雜度 $O((\log n)^D)$，D 代表維度。

7 參考

- https://hackmd.io/@wiwiho/cp-note/%2F%40wiwiho%2FCPN-segment-tree
- https://hackmd.io/@wiwiho/cp-note/%2F%40wiwiho%2FCPN-sparse-table
- https://hackmd.io/@wiwiho/cp-note/%2F%40wiwiho%2FCPN-binary-indexed-tree
- https://zh.wikipedia.org/zh-tw/%E7%B7%9A%E6%AE%B5%E6%A8%B9
- https://zerojudge.tw/ShowProblem?problemid=d539
- https://66lemon66.blogspot.com/2021/01/zerojudge-d539-max-c.html
- https://tioj.ck.tp.edu.tw/problems/1224
- http://cbdcoding.blogspot.com/2015/02/tioj-1224hoj-12.html
- https://zhuanlan.zhihu.com/p/103616664
- https://www.youtube.com/watch?v=GLuT4zKzdjc&t=1279s&ab_channel=sprout-tw
- https://www.youtube.com/watch?v=5DAIKf61xLs&ab_channel=sprout-tw
- https://blog.csdn.net/narcissus2_/article/details/89423591
- https://blog.csdn.net/iwts_24/article/details/81603603
- https://reurl.cc/XjpxLj
- https://reurl.cc/3oRyOO
- https://reurl.cc/b2qe7E