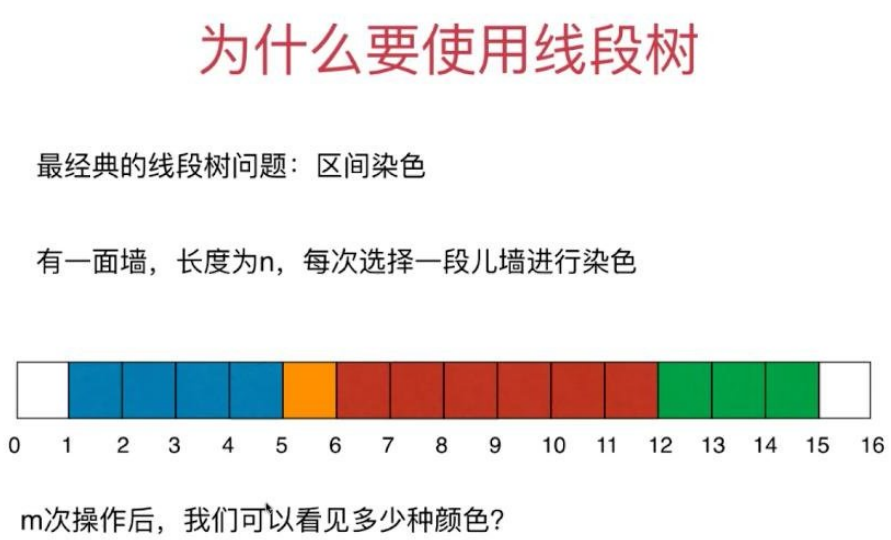
线段树（区间树）

什么是线段树

- 线段树可以用于区间相关的问题，如涂色，区间和等
    - 区间染色问题



对于给定的区间，进过m次的染色后我们能在区间`[i,j]`中看到多少种颜色呢?

- 第二类问题，区间查询问题

区间查询问题实际上是基于区间的统计查询问题，同时需要注意的是这里的**区间大小是固定的**，但是其中的数据是在动态的变化的，使用线段树可以帮助我们更好的解决这一问题。

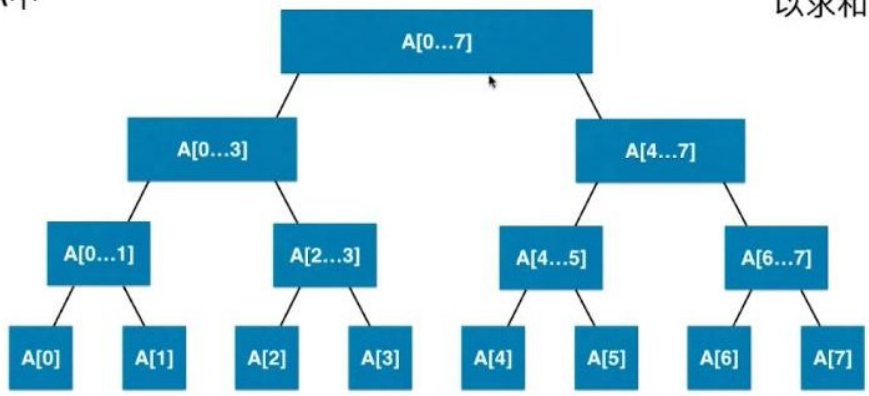
例如，我们查询2017年注册用户中消费最高的用户？消费最少的用户？学习时间最长的用户？或者是太空区间中天体总量？这些数据都不是静态的，都是在不断变化过程中的。

- 线段树的实现
    - 对于区间问题，一个朴素的想法就是使用数组来实现。但是线段树并不一定是完全二叉树，如果直接使用数组实现则不太方便，那我们的想法可以是**使用空间换取时间，将线段树人为的构造成满二叉树**。
    - 另一种思想则是使用链表构造线段树。
- 什么是线段树

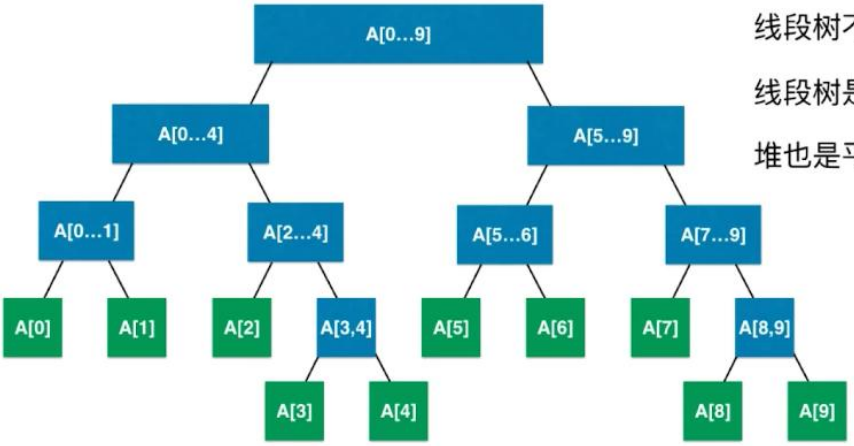
什么是线段树

在数组A中

以求和为例



什么是线段树

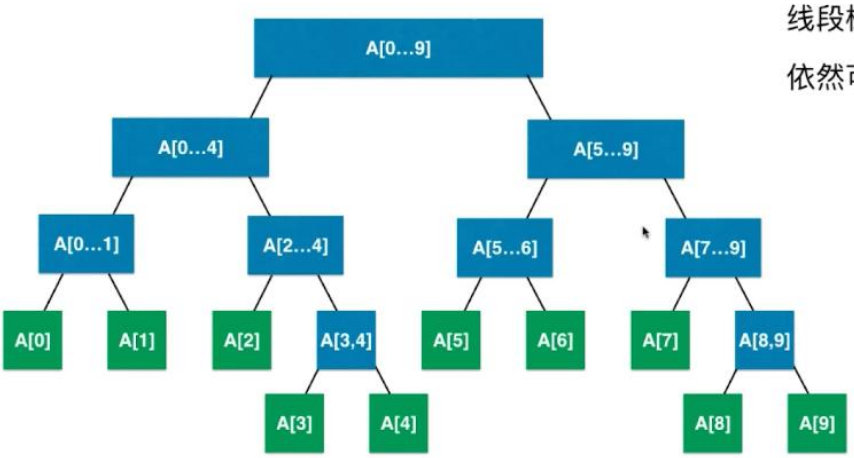


线段树不是完全二叉树

线段树是平衡二叉树

堆也是平衡二叉树

什么是线段树



线段树是平衡二叉树

依然可以用数组表示

看做满二叉树

什么是线段树

如果区间有n个元素 数组表示需要有多少节点?

0层：1

1层：2

2层：4

3层：8

…

h-1层：2^(h-1)

对满二叉树：

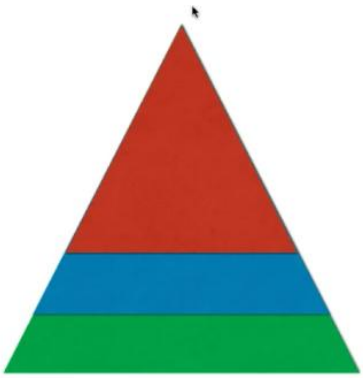
h层，一共有 2^h - 1 个节点（大约是2^h）

最后一层(h-1层)，有 2^(h-1)个节点

最后一层的节点数大致等于前面所有层节点之和

什么是线段树

如果区间有n个元素 数组表示需要有多少节点?



→ 如果n=2^k 只需要2n的空间

→ 最坏情况，如果n=2^k+1 需要4n的空间

什么是线段树

如果区间有n个元素 数组表示需要有多少节点?

需要4n的空间

我们的线段树不考虑添加元素，即区间固定

使用4n的静态空间即可

- 对于用数组表示的线段树，其**大小是固定的，我们不考虑线段树的添加和删除操作**
- 对于有**n**个数据的线段树，我们需要有**4n**大小的空间用来构建满二叉树
- 用数组表示线段树时，会存在较大的空间浪费，是一种**空间换取时间的做法**。

使用数组构建线段树

- 基本构建

```
public class SegementTree<E>{

    private Merger merger;
    //存储用户传进来的数据
    private E[] data;
    //用于构建树
    private E[] tree;

    public SegementTree(E[] arr, Merger merger){
        data = new (E [])Object[arr.length];
        this.merger = merger;

        for(int i=0;i<arr.length;i++)
            data[i] = arr[i];

        tree = new (E [])Object[4*arr.length];

        //构建线段树,注意根节点表示的范围是数据的范围
        buildSegementTree(0,0,arr.length-1);
    }

    //计算左孩子
    private int leftChild(int index){
        return 2*index + 1;
    }

    //计算右孩子
    private int rightChild(int index){
        return 2*index + 2;
    }

    //当前节点索引        treeIndex
    //当前节点的左边界      l
    //当前节点的右边界      r
    private void buildSegementTree(int treeIndex, int l, int r){
        //如果l==r,区间中只含有一个节点，即走到了叶子节点
        if( l == r){
            tree[treeIndex] = data[l];
            return ;
        }
        //如果区间还是一个范围，则需要继续划分为左孩子和右孩子，并将左右孩子处理到根节点
        //为什么是处理？ 可以是求和放到父节点，也可以是取最大值放到父节点，这需要根据我们
的需求而定

        // int mid = (l + r) / 2; 不要使用这种方式计算中值，可能会造成数据范围溢出
        int mid = l + (r - l) / 2;

        //计算左孩子和右孩子的索引
        int leftTreeIndex = leftChild(treeIndex);
        int rightTreeIndex = rightChild(treeIndex);

        //建立左子树
        buildSegementTree(leftTreeIndex, l, mid);
        //建立右子树
        buildSegementTree(rightTreeIndex, mid+1, r);
```

```
        //合并左右子树
        tree[treeIndex] =
merger.merge(tree[leftTreeIndex],tree[rightTreeIndex]);
    }

    public int getSize(){
        return data.length;
    }

    //我们可以通过索引获取叶节点的值，这也是我们为什么需要data数组的原因
    public E get(int index){
        if(index<0 || index>data.length-1)
            throw new IllegalArgumentError("index out of range!");
        return data[index];
    }
}
```

- 对于父节点表示的意义，如求和或者最大值等，需要用户自行定义，因此我们应当提供相应的接口。

```
public interface Merger<E>{
    E merger(E a, E b);
}
```

- 对于线段树一个很重要的操作就是获取区间的值，对于给定的区间`[queryL,queryR]`怎么获取其值呢?
    1. 我们需要从根节点开始寻找，给定根节点的索引为`treeIndex`，区间范围为`[l,r]`，计算区间中值为`mid = l + ( r- l) / 2`
    2. 如果要查询的区间在左子树中，即`queryR < = mid`，则在左子树中寻找
    3. 如果查询的区间在右子树中，即`queryL >= mid +1`，则在右子树中寻找
    4. 如果查询的区间包含左右两个子树即`queryL < mid < queryR`，则将其拆分为只在左右子树中的两个区间`[queryL,mind]`和`[mid+1,queryR]`
    5. 如果查询区间与树区间重合即`queryL == l and queryR == r`，则返回

```
public E query(int queryL, int queryR){
    if(queryL < 0 || queryR >= data.length || queryL > queryR)
        throw new IllegalArgumentError("index out of range!");
    return query(0,0,data.length-1,queryL,queryR);
}

//treeIndex 区间节点
//l         区间左边界
//r         区间右边界
//queryL    查询左边界
//queryR    查询右边界
private E query(int treeIndex, int l, int r, int queryL, int queryR){
    if(queryL == l && queryR == r){
        return tree[treeIndex];
    }

    int mid = l + (r - l) / 2;

    int leftTreeIndex = leftChild(treeIndex);
    int rightTreeIndex = rightChild(treeIndex);
```

```
    //查询边界落在左子树
    if(query <= mid)
        return query(leftTreeIndex,l,mid,queryL,queryR);
    //查询边界落在右子树
    if(queryL >= mid+1)
        return query(rightTreeIndex,mid+1,r,queryL,queryR);
    //查询结果落在左右两颗子树上
    E leftResult = query(leftTreeIndex,l,mid,queryL,mid);
    E rightResult = query(rightTreeIndex,mid+1,r,mid+1,queryR);

    return merger.merge(leftResult, rightResult);
}
```

- 如何将索引为`index`的位置的点的数据进行更新也是一个重点，当更新数据时会造成其父节点的值发生变化。

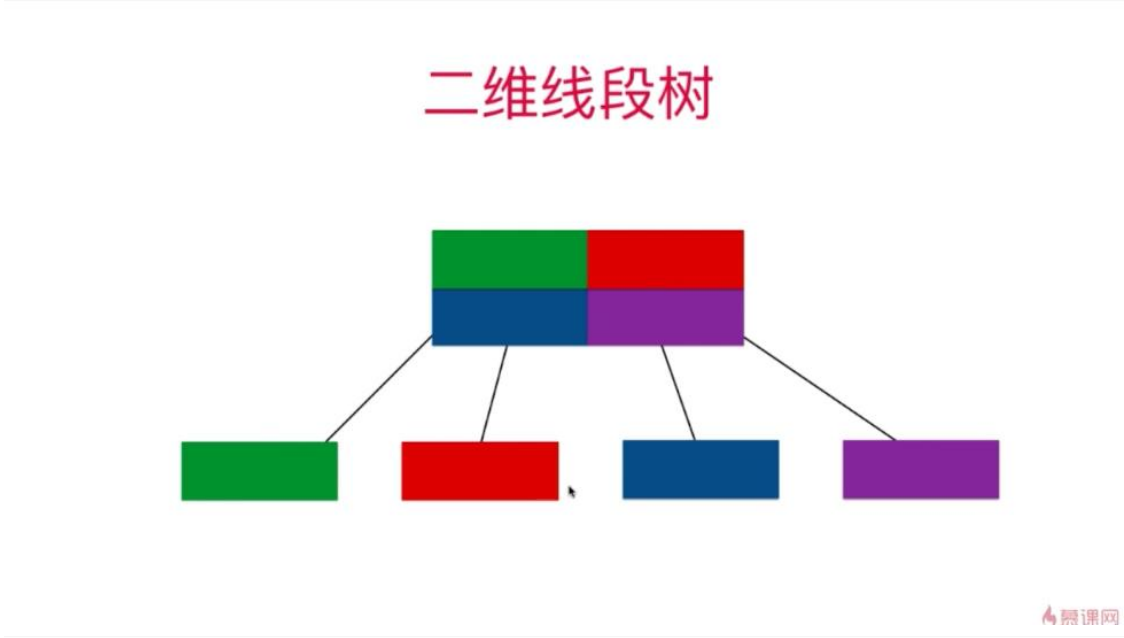
  此处，我们选择的方法是：

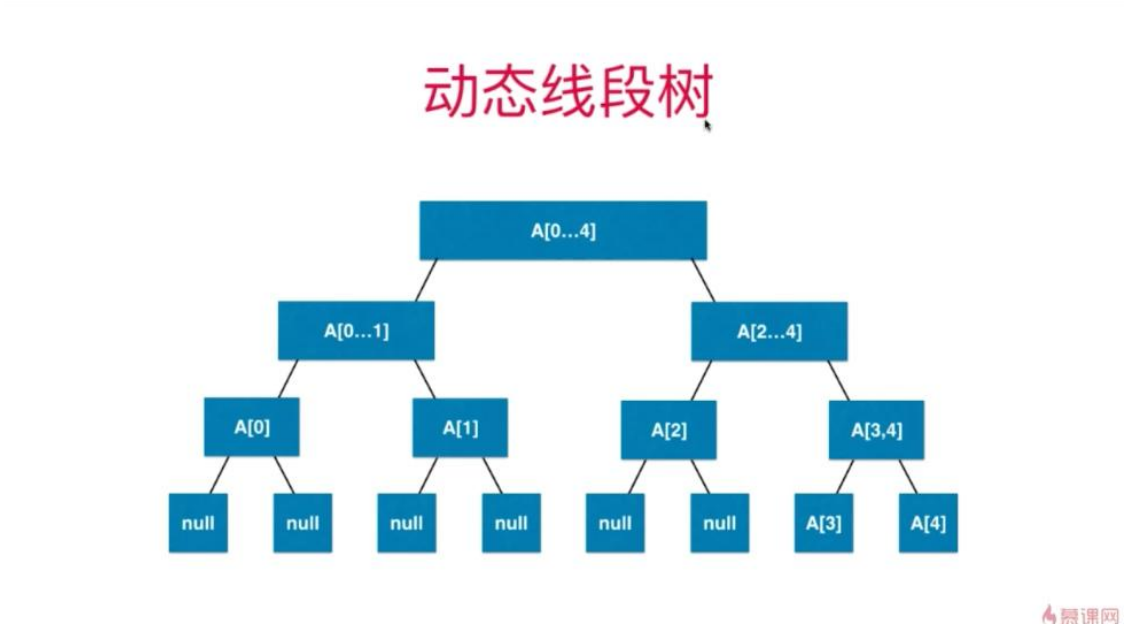
  1. 给定节点以及节点的区间
  2. 给定需要更新的数据索引，以及数据值
  3. 通过数据索引对节点区间不断的分割，直到`l == r == index`，则可以实现节点以及其父节点值的更新

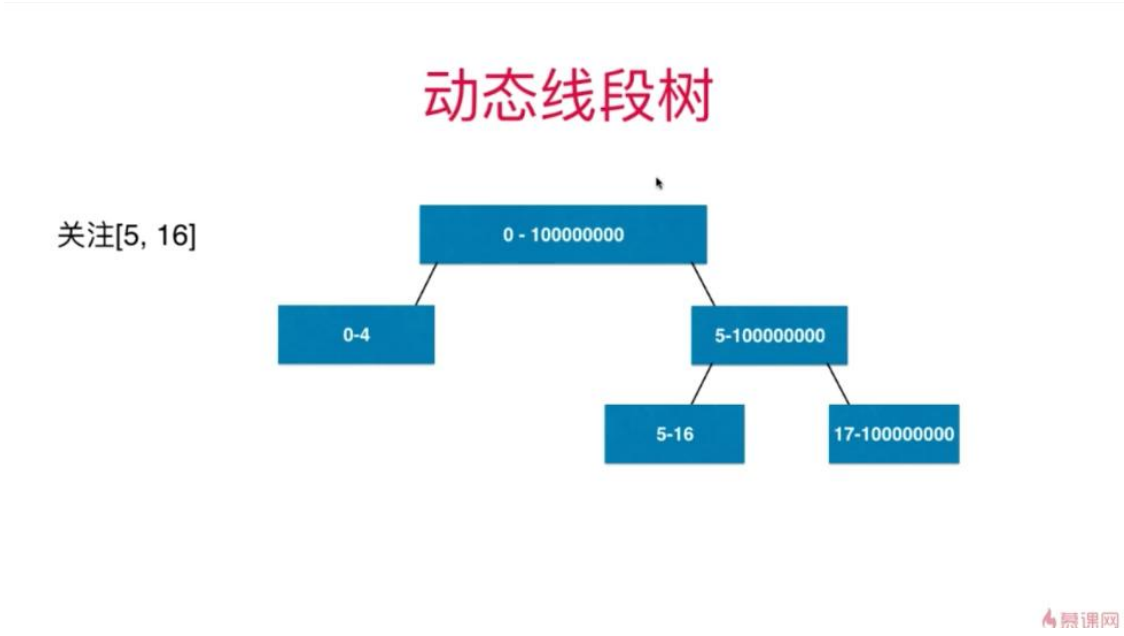
```
public void set(int index, E val){
    if(index < 0 || index >= data.length)
        throw new IllegalArgumentError("index out of range!");
    //记得要更新data数组中的元素值
    data[index] = e;
    set(0,0,data.length-1,index,val);
}

private void set(int treeIndex ,int l, int r, int index, E val){
    if(l == r){
        tree[treeIndex] = val;
        return;
    //为什么不考虑 l == r == index ?
    }

    int mid = l + (r - l) / 2;

    int leftTreeIndex = leftChild(treeIndex);
    int rightTreeIndex = rightChild(treeIndex);

    if(index <= mid)
        set(leftTreeIndex,l,mid,index,val);
    else
        set(rightTreeIndex,mid+1,r,index,val);
    //是否已经结束了?
    //并没有，此时已经更新过了节点的值，但是对于节点的父节点的值还是没有更新的
    tree[treeIndex] =
merger.merge(tree[leftTreeIndex],tree[rightTreeIndex]);
    //如果不加上述这句，最后只会更新tree中的某一个节点，而节点的父节点都没有更新。
}
```

- 如何更新线段树的区间呢，可以借助与懒更新的思想，此处不在给出
- 线段树属于高级数据结构，了解其基本用法即可

扩展的线段树

- 

- 

- 

- 练习题
  - LeetCode 303

- LeetCode 307