算法分析与设计

第二讲线段树

纪洪波

通化师范学院　计算机学院

2016 年 9 月 4 日

目录

问题描述

老管家是一个聪明能干的人。他为财主工作了整整 10 年，财主为了让自已账目更加清楚。要求管家每天记 k 次账，由于管家聪明能干，因而管家总是让财主十分满意。但是由于一些人的挑拨，财主还是对管家产生了怀疑。于是他决定用一种特别的方法来判断管家的忠诚，他把每次的账目按 $1,\ 2,\ 3, \cdots$ 编号，然后不定时的问管家问题，问题是这样的：在 a 到 b 号账中最少的一笔是多少？为了让管家没时间作假他总是一次问多个问题。

问题输入输出描述

输入格式

输入中第一行有两个数 m, n 表示有 $m(m \leq 100000)$ 账,n 表示有 n 个问题,$n \leq 100000$。

第二行为 m 个数, 分别是账目的钱数

后面 n 行分别是 n 个问题, 每行有 2 个数字说明开始结束的账目编号。

输出格式

输出文件中为每个问题的答案。具体查看样例。

问题输入输出样例

输入

10 3

1 2 3 4 5 6 7 8 9 10

2 7

3 9

1 10

输出

2 3 1

怎么梦好呢?

如果我有一个函数 $y = f(l, r)$,每当输入一个区间 $[l, r]$,则会返回该区间内所有数的最小值。

这个梦想非常好实现的！同学们现在的能力就可以：

```
f(int l,int r){
  int min=+infty;//infty 代表最大值
  for(int i=l;i<=r;++i){
    if(min>a[i])min=a[i];
  }
  return min;
}
```

请同学们思考,这段代码能不能达到目的,还有什么问题?

第一个梦碎了!!

好多的重复计算! 大家想想看，能不能举出重复计算的例子?

答案：

记忆化!!!!

第一个梦碎了!!

好多的重复计算！大家想想看，能不能举出重复计算的例子？

答案：

记忆化!!!!

第一个梦碎了!!

好多的重复计算！大家想想看，能不能举出重复计算的例子？

答案：

记忆化!!!!

第二个梦

如果我有一个函数 $y = f(l, r)$,每当输入一个区间 $[l, r]$,则会返回该区间内所有数的最小值。

加入记忆化的成分:

```
f(int l,int r){
  if(vis[l][r]) return vis[l][r];//全部初始化无穷大
  int min=+infty;//infty 代表最大值
  for(int i=l;i<=r;++i){
    if(min>a[i])min=a[i];
  }
  vis[l][r]=min;
  return min;
}
```

请同学们思考,这段代码能不能达到目的,还有什么问题?

第二个梦碎了!!

好大的数组呀! 大家想想看,那个数组能有多大?

答案:

vis 数组!!!!

$$n \times n$$

第二个梦碎了!!

好大的数组呀! 大家想想看, 那个数组能有多大?

答案:

vis 数组!!!!

$$n \times n$$

第二个梦碎了!!

好大的数组呀! 大家想想看,那个数组能有多大?

答案:

vis 数组!!!!

$$n \times n$$

换一个思路!!

我们可以将存储量减小一些,改变一下数据的组织。

如下:

$$\underbrace{\underbrace{\underbrace{1,\ 2}_{f(1,2)}\ \underbrace{3,\ 4,\ 5}_{f(3,5)}}_{f(1,5)}\ \underbrace{\underbrace{6,\ 7}_{f(6,7)}\ \underbrace{8,\ 9,\ 10}_{f(8,10)}}_{f(6,10)}}_{f(1,10)}$$

当然具体实现还需要继续细致的思考一下。
如果你问我 $f(1,\ 10)$,我可以直接给你答案。
如果你问我 $f(3,\ 10)$? 我可以给你 $f(3,\ 5)+f(6,\ 10)$。

想象力! 上面的图,像什么?

换一个思路!!

我们可以将存储量减小一些,改变一下数据的组织。

如下:

$$\underbrace{\underbrace{\underbrace{1,\ 2}_{f(1,2)}\ \underbrace{3,\ 4,\ 5}_{f(3,5)}}_{f(1,5)}\ \underbrace{\underbrace{6,\ 7}_{f(6,7)}\ \underbrace{8,\ 9,\ 10}_{f(8,10)}}_{f(6,10)}}_{f(1,10)}$$

当然具体实现还需要继续细致的思考一下。
如果你问我 $f(1,\ 10)$,我可以直接给你答案。
如果你问我 $f(3,\ 10)$? 我可以给你 $f(3,\ 5)+f(6,\ 10)$。

想象力!上面的图,像什么?

换一个思路!!

我们可以将存储量减小一些,改变一下数据的组织。

如下:

$$\underbrace{\underbrace{\underbrace{1,\ 2}_{f(1,2)}\ \underbrace{3,\ 4,\ 5}_{f(3,5)}}_{f(1,5)}\ \underbrace{\underbrace{6,\ 7}_{f(6,7)}\ \underbrace{8,\ 9,\ 10}_{f(8,10)}}_{f(6,10)}}_{f(1,10)}$$

当然具体实现还需要继续细致的思考一下。
如果你问我 $f(1,\ 10)$,我可以直接给你答案。
如果你问我 $f(3,\ 10)$? 我可以给你 $f(3,\ 5)+f(6,\ 10)$。

想象力！上面的图，像什么？

换一个思路!!

我们可以将存储量减小一些,改变一下数据的组织。

如下:

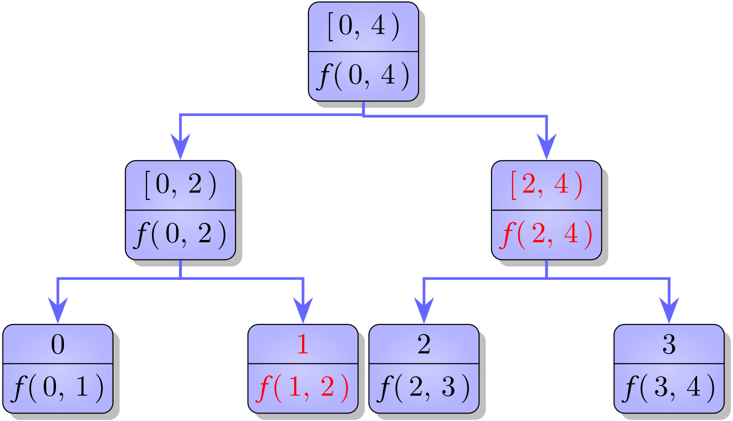
$$\underbrace{\underbrace{\underbrace{1,\ 2}_{f(1,2)}\ \underbrace{3,\ 4,\ 5}_{f(3,5)}}_{f(1,5)}\ \underbrace{\underbrace{6,\ 7}_{f(6,7)}\ \underbrace{8,\ 9,\ 10}_{f(8,10)}}_{f(6,10)}}_{f(1,10)}$$

当然具体实现还需要继续细致的思考一下。
如果你问我 $f(1,\ 10)$,我可以直接给你答案。
如果你问我 $f(3,\ 10)$? 我可以给你 $f(3,\ 5) + f(6,\ 10)$。

想象力! 上面的图,像什么?

最终答案

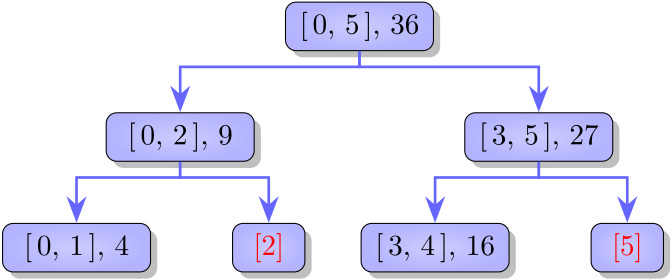
好吧！谜底揭开如下图（我们少用几个元素）：
如何计算 $[1, 4)$ 的元素和？



实例

数据数组 $arr = \{1, 3, 5, 7, 9, 11\}$,注意数组下标从 1 算起!

建成的线段树如下图(第四层 4 个节点略):



线段树数组表示为 $st = \{36, 9, 27, 4, 5, 16, 11, 1, 3, \wedge, \wedge, 7, 9, \wedge, \wedge\}$,

因为是从下标 0 开始存储,所以下标为 i 的节点的左右子节点分别为?

$2 \times i$ 和 $2 \times i + 1$

注意 2 和 5 红色节点其实不存在,因为已经没有区间了。

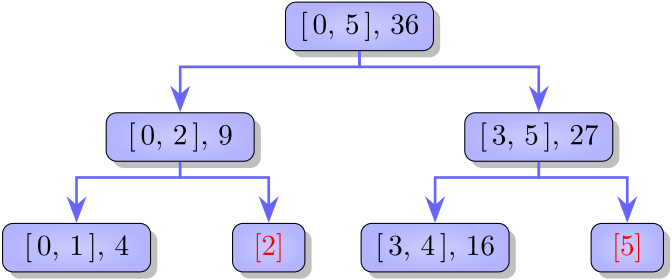
注意这个例子用的是左右闭区间,真的没问题吗?

理解原理就没问题!下面来研究怎么构建这棵线段树!

实例

数据数组 $arr = \{1, 3, 5, 7, 9, 11\}$,注意数组下标从 1 算起!

建成的线段树如下图(第四层 4 个节点略):



线段树数组表示为 $st = \{36, 9, 27, 4, 5, 16, 11, 1, 3, \wedge, \wedge, 7, 9, \wedge, \wedge\}$,

因为是从下标 0 开始存储,所以下标为 i 的节点的左右子节点分别为?

$2 \times i$ 和 $2 \times i + 1$

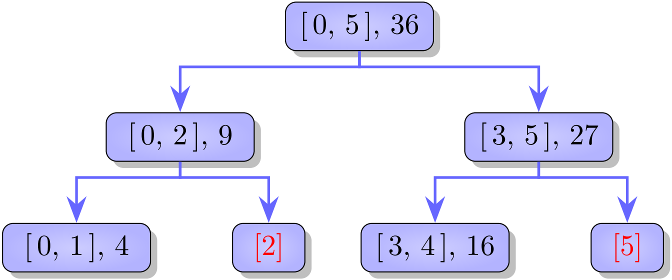
注意 2 和 5 红色节点其实不存在,因为已经没有区间了。

注意这个例子用的是左右闭区间,真的没问题吗?

理解原理就没问题!下面来研究怎么构建这棵线段树!

实例

数据数组 $arr = \{1, 3, 5, 7, 9, 11\}$,注意数组下标从 1 算起!
建成的线段树如下图(第四层 4 个节点略):



线段树数组表示为 $st = \{36, 9, 27, 4, 5, 16, 11, 1, 3, \wedge, \wedge, 7, 9, \wedge, \wedge\}$,

因为是从下标 0 开始存储,所以下标为 i 的节点的左右子节点分别为?

$2 \times i$ 和 $2 \times i + 1$

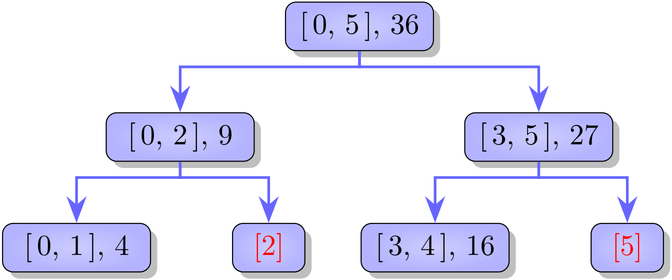
注意 2 和 5 红色节点其实不存在,因为已经没有区间了。

注意这个例子用的是左右闭区间,真的没问题吗?

理解原理就没问题!下面来研究怎么构建这棵线段树!

实例

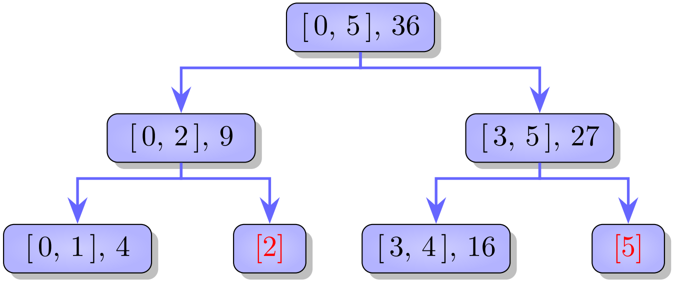
数据数组 $arr = \{1, 3, 5, 7, 9, 11\}$,注意数组下标从 1 算起!
建成的线段树如下图(第四层 4 个节点略):



线段树数组表示为 $st = \{36, 9, 27, 4, 5, 16, 11, 1, 3, \wedge, \wedge, 7, 9, \wedge, \wedge\}$,
因为是从下标 0 开始存储,所以下标为 i 的节点的左右子节点分别为?
$2 \times i$ 和 $2 \times i + 1$
注意 2 和 5 红色节点其实不存在,因为已经没有区间了。
注意这个例子用的是左右闭区间,真的没问题吗?
理解原理就没问题!下面来研究怎么构建这棵线段树!

实例

数据数组 $arr=\{1,3,5,7,9,11\}$,注意数组下标从 1 算起!
建成的线段树如下图(第四层 4 个节点略):



线段树数组表示为 $st=\{36,9,27,4,5,16,11,1,3,\wedge,\wedge,7,9,\wedge,\wedge\}$
因为是从下标 0 开始存储,所以下标为 i 的节点的左右子节点分别为?
$2\times i$ 和 $2\times i+1$
注意 2 和 5 红色节点其实不存在,因为已经没有区间了。
注意这个例子用的是左右闭区间,真的没问题吗?
理解原理就没问题!下面来研究怎么构建这棵线段树!

问题

到底需要多大的数组存储线段树?

有些人写节点数不超过 $2n-1$,对吗? 为什么?

对! $\begin{cases} 2\times n_2+n_1 = N-1 & \cdots\cdots ① \\ n_2+n_1+n_0 = N & \cdots\cdots ② \end{cases}$

② − ① 得: $n_0 - n_2 = 1$ 即 $n_0 = n_2 - 1$。

那么可以申请 $2n-1$ 这么大的数组存储线段树?

不行,前面的例子就不行,因为中间有空着的元素! 申请 $2n-1$ 不够大! 怎么办?

灵机一动!

深度 $k = \lceil ln(2n-1) \rceil$

所以线段树的大小为 $2^{\lceil ln(2n-1) \rceil} - 1$ 就足够了!

实际写代码的时候,我们不这样计算的,好傻! 应该怎么做? 看后面!

问题

到底需要多大的数组存储线段树?
有些人写节点数不超过 $2n-1$,对吗? 为什么?

对! $\begin{cases} 2\times n_2 + n_1 = N-1 & \cdots\cdots ① \\ n_2 + n_1 + n_0 = N & \cdots\cdots ② \end{cases}$

② − ① 得: $n_0 - n_2 = 1$ 即 $n_0 = n_2 - 1$。
那么可以申请 $2n-1$ 这么大的数组存储线段树?
不行,前面的例子就不行,因为中间有空着的元素! 申请 $2n-1$ 不够大! 怎么办?

灵机一动!

深度 $k = \lceil ln(2n-1) \rceil$
所以线段树的大小为 $2^{\lceil ln(2n-1) \rceil} - 1$ 就足够了!
实际写代码的时候,我们不这样计算的,好傻! 应该怎么做? 看后面!

问题

到底需要多大的数组存储线段树?
有些人写节点数不超过 $2n-1$,对吗? 为什么?

$$\text{对!}\begin{cases} 2\times n_2+n_1 &= N-1 & \cdots\cdots \text{❶} \\ n_2+n_1+n_0 &= N & \cdots\cdots \text{❷} \end{cases}$$

❷ − ❶ 得:$n_0-n_2=1$ 即 $n_0=n_2-1$。
那么可以申请 $2n-1$ 这么大的数组存储线段树?
不行,前面的例子就不行,因为中间有空着的元素! 申请 $2n-1$ 不够大! 怎么办?

灵机一动!

深度 $k=\lceil ln(2n-1)\rceil$
所以线段树的大小为 $2^{\lceil ln(2n-1)\rceil}-1$ 就足够了!
实际写代码的时候,我们不这样计算的,好傻! 应该怎么做? 看后面!

问题

到底需要多大的数组存储线段树?
有些人写节点数不超过 $2n-1$,对吗? 为什么?

对!
$$\begin{cases} 2\times n_2+n_1 &= N-1 \quad \cdots\cdots ❶ \\ n_2+n_1+n_0 &= N \quad \cdots\cdots ❷ \end{cases}$$

❷ − ❶ 得:$n_0-n_2=1$ 即 $n_0=n_2-1$。
那么可以申请 $2n-1$ 这么大的数组存储线段树?
不行,前面的例子就不行,因为中间有空着的元素! 申请 $2n-1$ 不够大! 怎么办?

灵机一动!

深度 $k=\lceil ln(2n-1)\rceil$
所以线段树的大小为 $2^{\lceil ln(2n-1)\rceil}-1$ 就足够了!
实际写代码的时候,我们不这样计算的,好傻! 应该怎么做? 看后面!

问题

到底需要多大的数组存储线段树?

有些人写节点数不超过 $2n-1$,对吗? 为什么?

对! $\begin{cases} 2\times n_2+n_1 & =N-1 & \cdots\cdots❶ \\ n_2+n_1+n_0 & =N & \cdots\cdots❷ \end{cases}$

❷ − ❶ 得:$n_0-n_2=1$ 即 $n_0=n_2-1$。

那么可以申请 $2n-1$ 这么大的数组存储线段树?

不行,前面的例子就不行,因为中间有空着的元素! 申请 $2n-1$ 不够大! 怎么办?

灵机一动!

深度 $k=\lceil ln(2n-1)\rceil$

所以线段树的大小为 $2^{\lceil ln(2n-1)\rceil}-1$ 就足够了!

实际写代码的时候,我们不这样计算的,好傻! 应该怎么做? 看后面!

数学归纳法

在动手写代码实现这个线段树之前，让我们花点儿时间用数学的方式重新审视我们的思维过程！

有没有更简洁，更显而易见且可以重复的思维过程？

数学归纳法！

同学们还记得第一节课讲的“分治归纳法”？如何求 $f(l, r)$？

同学们思考一下，我们按什么归纳？这里没有 n。我们按照 l？还是 r 归纳？

灵机一动！（随时准备着用数学的武器武装自己！）

[illegible]

归纳假设：已知如何找到 [illegible] 和 [illegible] 区间的最小值。

归纳基础 $n = 1$ 是显而易见的。

数学归纳法

在动手写代码实现这个线段树之前，让我们花点儿时间用数学的方式重新审视我们的思维过程！

有没有更简洁，更显而易见且可以重复的思维过程？

数学归纳法！

同学们还记得第一节课讲的“分治归纳法”？如何求 $f(l, r)$？

同学们思考一下，我们按什么归纳？这里没有 n。我们按照 l？还是 r 归纳？

灵机一动！（随时准备着用数学的武器武装自己！）

数学归纳法

在动手写代码实现这个线段树之前,让我们花点儿时间用数学的方式重新审视我们的思维过程!

有没有更简洁,更显而易见且可以重复的思维过程?

数学归纳法!

同学们还记得第一节课讲的"分治归纳法"? 如何求 $f(l, r)$?

同学们思考一下,我们按什么归纳? 这里没有 n。我们按照 l? 还是 r 归纳?

灵机一动!(随时准备着用数学的武器武装自己!)

归纳假设:已知如何找到 $\left[l, l+\frac{n}{2}\right]$ 和 $\left[l+\frac{n}{2}, r\right]$ 区间的最小值;

归纳基础 $n=1$ 是显而易见的。

数学归纳法

在动手写代码实现这个线段树之前，让我们花点儿时间用数学的方式重新审视我们的思维过程！

有没有更简洁，更显而易见且可以重复的思维过程？

数学归纳法！

同学们还记得第一节课讲的“分治归纳法”？如何求 $f(l, r)$?

同学们思考一下，我们按什么归纳？这里没有 n。我们按照 l？还是 r 归纳？

灵机一动！（随时准备着用数学的武器武装自己！）

数学归纳法

在动手写代码实现这个线段树之前，让我们花点儿时间用数学的方式重新审视我们的思维过程！

有没有更简洁，更显而易见且可以重复的思维过程？

数学归纳法！

同学们还记得第一节课讲的“分治归纳法”？如何求 $f(l, r)$？

同学们思考一下，我们按什么归纳？这里没有 n。我们按照 l？还是 r 归纳？

数学归纳法

在动手写代码实现这个线段树之前，让我们花点儿时间用数学的方式重新审视我们的思维过程！

有没有更简洁，更显而易见且可以重复的思维过程？

数学归纳法！

同学们还记得第一节课讲的“分治归纳法”？如何求 $f(l, r)$？

同学们思考一下，我们按什么归纳？这里没有 n。我们按照 l？还是 r 归纳？

灵机一动！（随时准备着用数学的武器武装自己！）

$$n = l - r$$

归纳假设：已知如何找到 $\left[l, l+\dfrac{n}{2}\right)$ 和 $\left[l+\dfrac{n}{2}, r\right]$ 区间的最小值。

归纳基础 $n = 1$ 是显而易见的。

数学归纳法

在动手写代码实现这个线段树之前，让我们花点儿时间用数学的方式重新审视我们的思维过程！

有没有更简洁，更显而易见且可以重复的思维过程？

数学归纳法！

同学们还记得第一节课讲的“分治归纳法”？如何求 $f(l, r)$？

同学们思考一下，我们按什么归纳？这里没有 n。我们按照 l？还是 r 归纳？

灵机一动！（随时准备着用数学的武器武装自己！）

$$n = l - r$$

归纳假设：已知如何找到 $\left[l,\ l+\dfrac{n}{2}\right)$ 和 $\left[l+\dfrac{n}{2},\ r\right]$ 区间的最小值。

归纳基础 $n = 1$ 是显而易见的。

数学归纳法

在动手写代码实现这个线段树之前，让我们花点儿时间用数学的方式重新审视我们的思维过程！

有没有更简洁，更显而易见且可以重复的思维过程？

数学归纳法！

同学们还记得第一节课讲的“分治归纳法”？如何求 $f(l, r)$?

同学们思考一下，我们按什么归纳？这里没有 n。我们按照 l? 还是 r 归纳？

灵机一动！（随时准备着用数学的武器武装自己！）

$$n = l - r$$

归纳假设：已知如何找到 $\left[l, \; l + \dfrac{n}{2}\right)$ 和 $\left[l + \dfrac{n}{2}, \; r\right]$ 区间的最小值。

归纳基础 $n = 1$ 是显而易见的。

构建线段树

```
//取中位数
int getMid(int s, int e){ return s+(e-s)/2;}
/* 构建线段树,主要是调用递归函数 buildSTRecall 完成 */
int *buildST(int arr[], int n){
  int _2power=1;
  int N=n*2;
  while(_2power<N){
    _2power=_2power<<1;
  }
  // 分配内存
  int *st = new int[_2power];
  // 构建线段树， 数据数组,L,R, 线段树， 根下标
  buildSTRecall(arr, 0, n-1, st, 0);
  return st;
}
```

构建线段树 II

```
// 递归来构建区间为 [ss..se] 的线段树 (st[si])
// si 当前节点在线段树中的下标
// return: 当前区间的总和
int buildSTRecall(int arr[], int ss, int se
                            , int *st, int si){
// 如果只有一个元素,说明到达了叶子节点
  if (ss == se){
    st[si] = arr[ss];  return arr[ss];
  }
// 递归的构建左右区间 (子线段树)
  int mid = getMid(ss, se);
 st[si] = buildSTRecall(arr, ss, mid, st, si*2+1)
   +buildSTRecall(arr, mid+1, se, st, si*2+2);
  return st[si];
}
```

线段树查询区间和

```
//查询,主要是调用 getSumRecall
int getSum(int *st, int n, int qs, int qe){
  if (qs < 0 || qe > n-1 || qs > qe)  {
    printf("Invalid Input");
    return -1;
  }
  return getSumRecall(st, 0, n-1, qs, qe, 0);
}
```

线段树查询区间和 II

```
/* st--> 线段树的指针
index--> 当前节点在线段树中的下标. 初始为 0
ss & se--> 当前节点 st[index] 做表示的区间 [ss->se]
qs & qe--> 要查询的区间 起始坐标 */
int getSumRecall(int *st, int ss, int se,
             int qs, int qe, int index){
// 如果查询的区间属于当前节点表示的区间
  if (qs <= ss && qe >= se) return st[index];
  // 完全不属于
  if (se < qs || ss > qe)  return 0;
  // 部分属于
  int mid = getMid(ss, se);
  return getSumRecall(st,ss,mid,qs,qe,2*index+1)
  +getSumRecall(st,mid+1,se,qs,qe,2*index+2);
}
```

线段树更新区间元素 I

```
// 更新 i 为新的值 new_val,
//主要是调用 updateValue
void update(int arr[], int *st,
            int n, int i, int new_val){
  if (i < 0 || i > n-1){
    printf("Invalid Input");
    return;
  }

  int diff = new_val - arr[i];
  arr[i] = new_val;
  // 更新线段树
  updateValue(st,0,n-1,i,diff,0);
}
```

线段树更新区间元素 II

```
/* st, index, ss 和 se 和函数 getSumRecall 相同
i--> 要更新的元素下标.(原始数组中的下标)
diff--> 需要增加的值。所有包含 i 的区间都会更新 */
void updateValue(int *st, int ss,
           int se, int i, int diff, int index){
  if (i < ss || i > se)  return;
  st[index] = st[index] + diff;
  if (se != ss)  {
    int mid = getMid(ss, se);
      updateValue(st,ss,mid,i,diff,2*index+1);
      updateValue(st,mid+1,se,i,diff,2*index+2);
  }
}
```

主函数

```c
#include <stdio.h>
#include <math.h>
//各个功能函数
int main(){
  int arr[] = {1, 3, 5, 7, 9, 11};
  int n = sizeof(arr)/sizeof(arr[0]);
  // 构建线段树
  int *st = buildST(arr, n);
  printf("Sum = %d\n", getSum(st, n, 1, 3));
  // 更新: arr[1] = 10 and 更新线段树相应区间
  update(arr, st, n, 1, 10);
  printf("Updated sum  = %d\n",
  getSum(st, n, 1, 3));
  return 0;
}
```

加油!!!

同学们课后要努力!!!!
将讲义中代码组合并运行!!!!