算法分析与设计

第三讲线段树-改进

纪洪波

通化师范学院　计算机学院

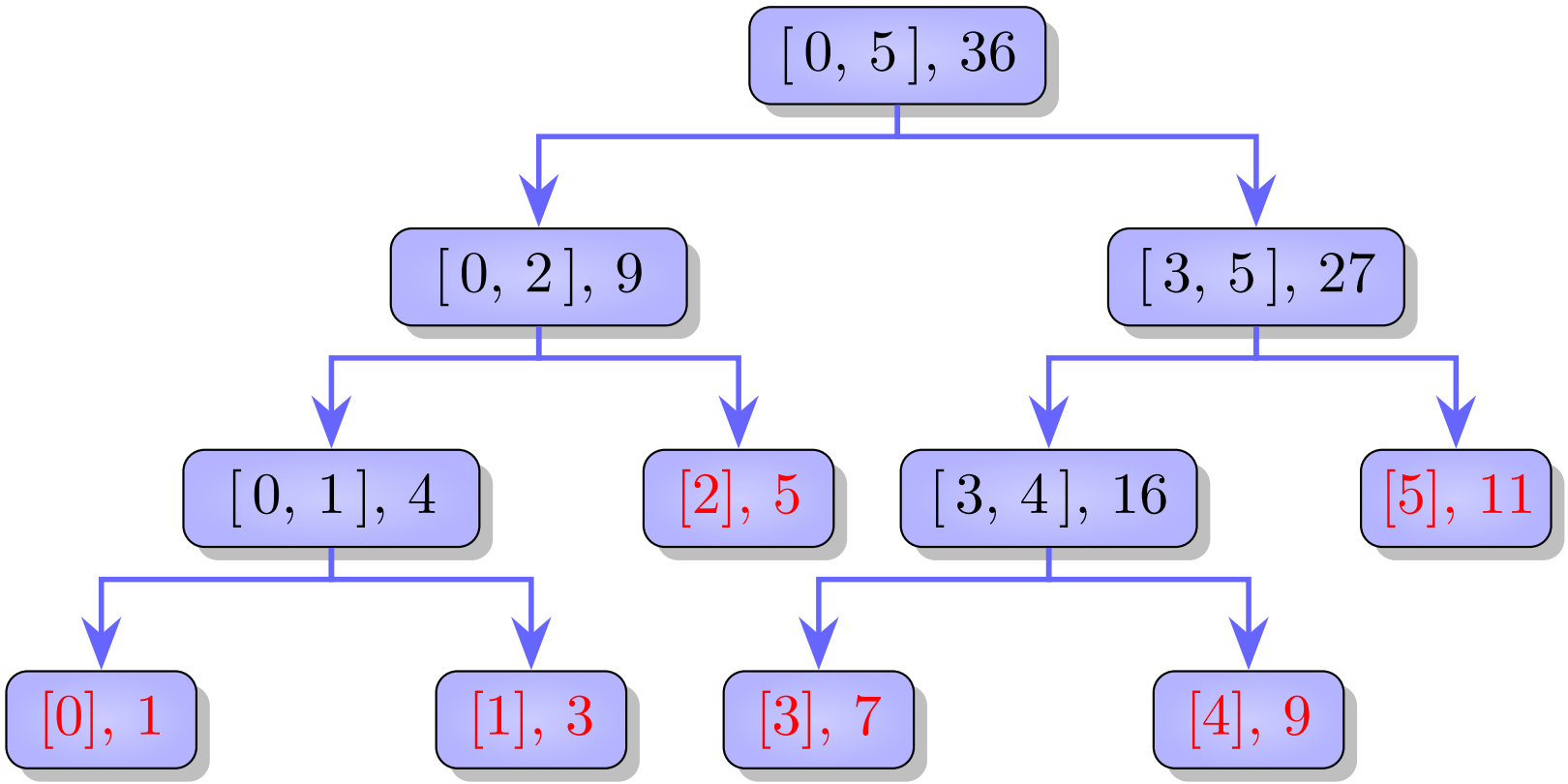
2016 年 9 月 4 日

目录

上次课实例

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：



线段树数组表示为

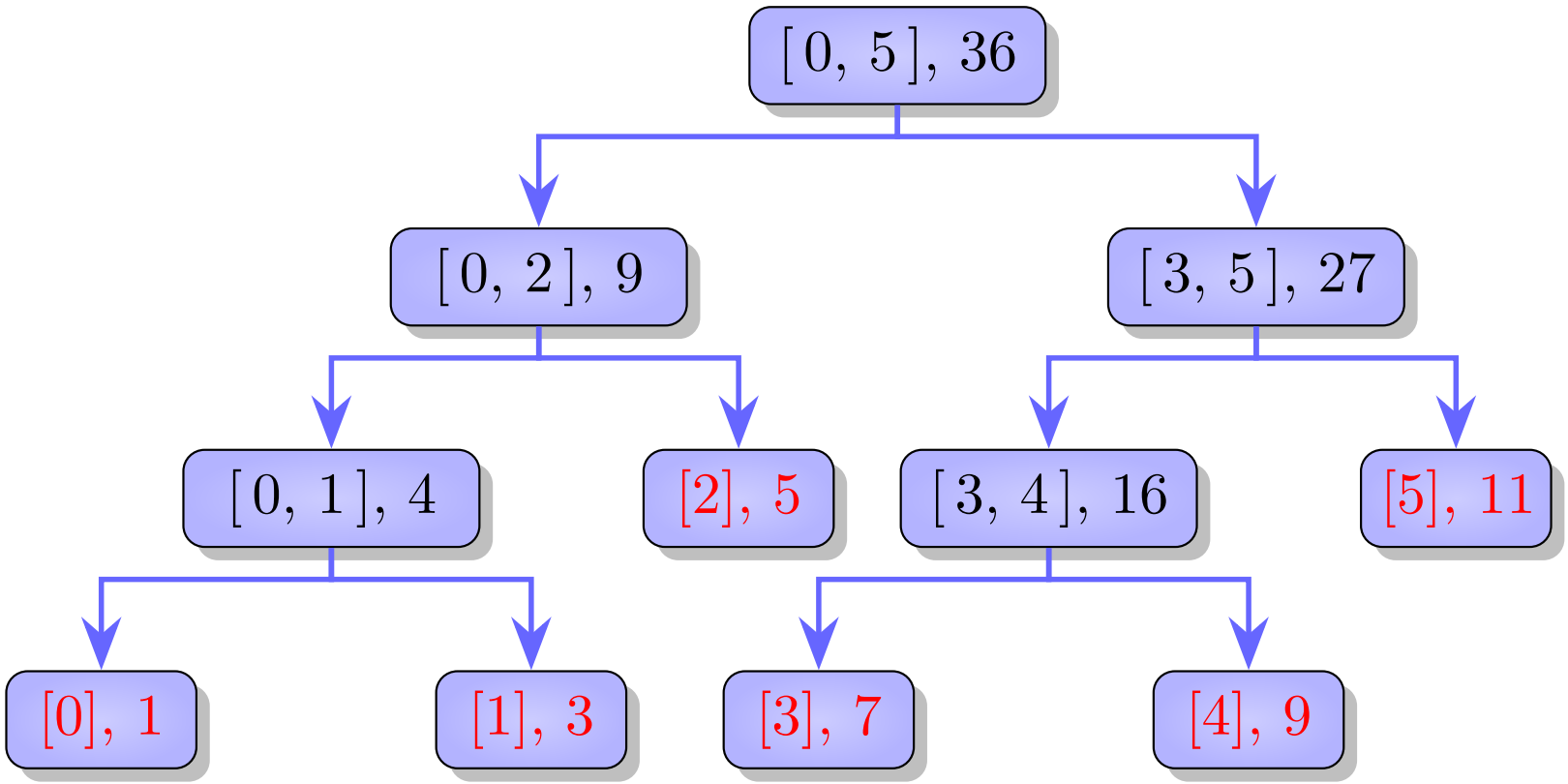
$st = \{36, 9, 27, 4, 5, 16, 11, 1, 3, \wedge, \wedge, 7, 9, \wedge, \wedge\}$，

这棵树难看吗？

上次课实例

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：



线段树数组表示为

$st = \{36, 9, 27, 4, 5, 16, 11, 1, 3, \wedge, \wedge, 7, 9, \wedge, \wedge\}$，

这棵树难看吗？

问题

看看的原因：

1. 有缺失分支。
2. 存储空间不好算。(只能按满二叉树算存储空间)

如果这棵二叉树是完全二叉树就好了，没有缺失的分支！

问题

看看的原因：

1. 有缺失分支。
2. 存储空间不好算。(只能按满二叉树算存储空间)

如果这棵二叉树是完全二叉树就好了，没有缺失的分支！

问题

看看的原因：

1. 有缺失分支。
2. 存储空间不好算。(只能按满二叉树算存储空间)

如果这棵二叉树是完全二叉树就好了，没有缺失的分支！

问题

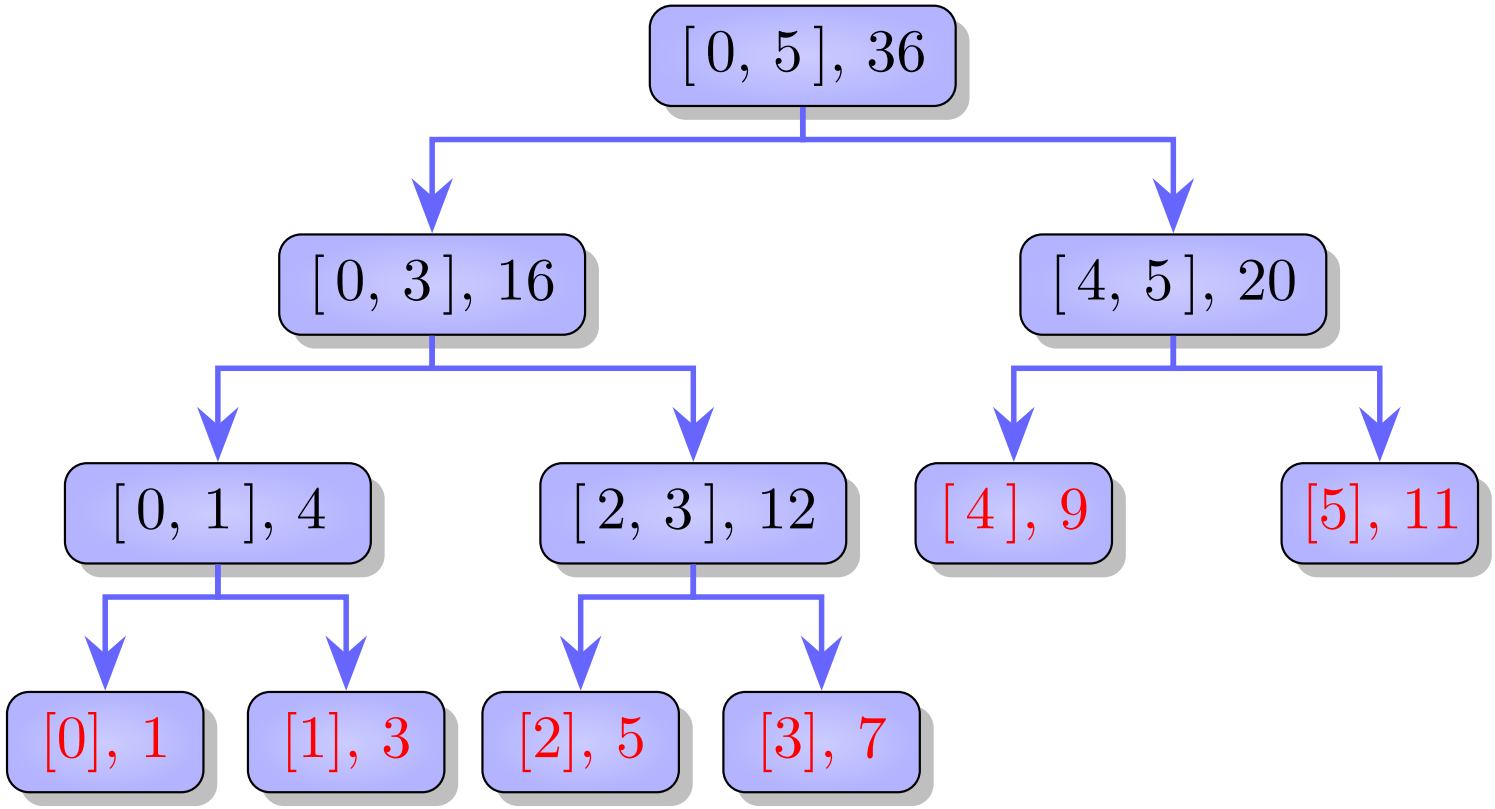
看看的原因：

1. 有缺失分支。
2. 存储空间不好算。(只能按满二叉树算存储空间)

如果这棵二叉树是完全二叉树就好了，没有缺失的分支！

我们的新梦想

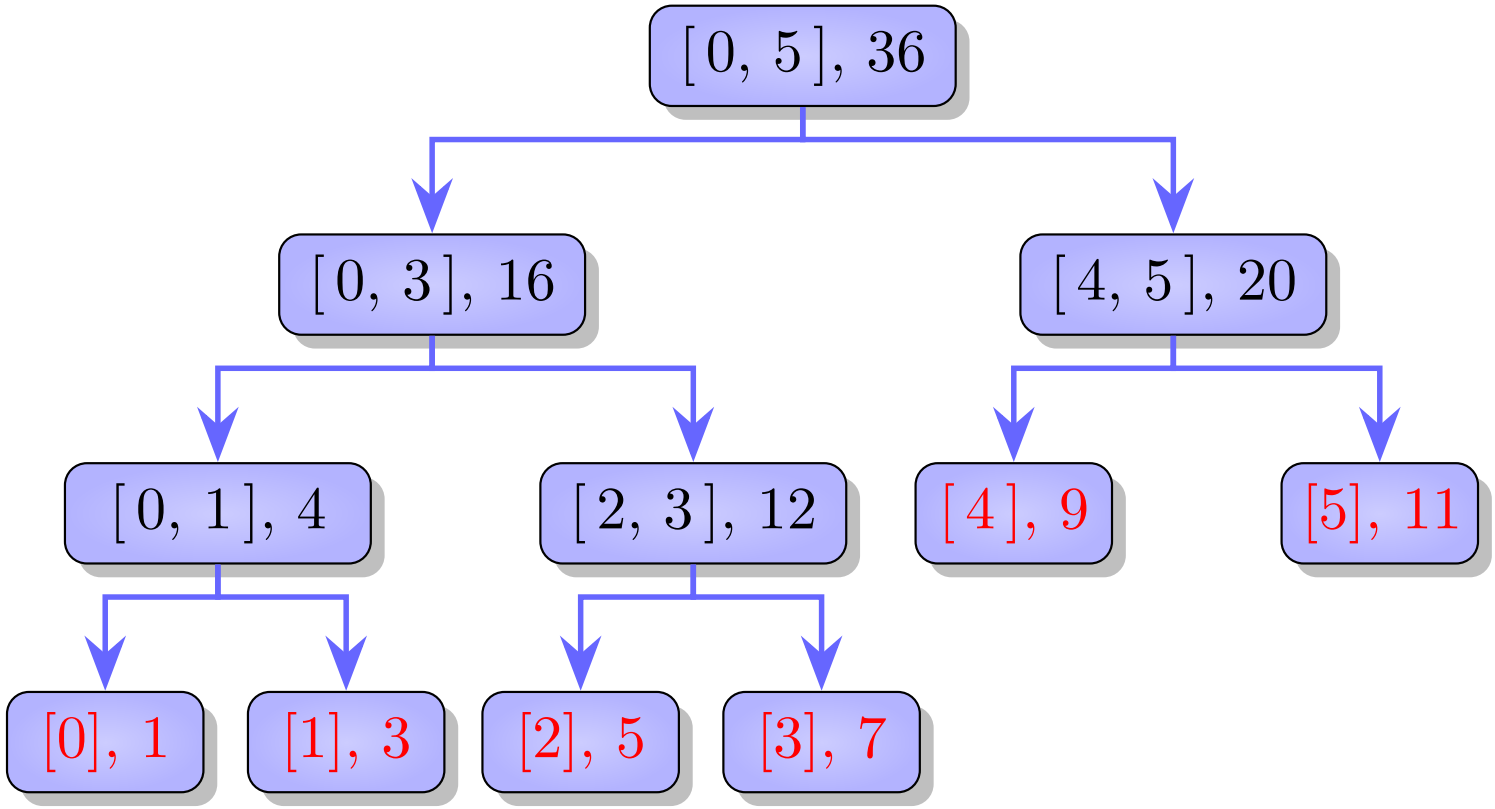
数据数组 $arr = \{1, 3, 5, 7, 9, 11\}$，注意数组下标从 1 算起！
建成的完美线段树如下图：



线段树数组表示为 $st = \{36, 16, 20, 4, 12, 9, 11, 1, 3, 7, 9\}$，
这棵树好看了，完美了！那么线段树数组大小是多少？
$2 \times n - 1$

我们的新梦想

数据数组 $arr = \{1, 3, 5, 7, 9, 11\}$，注意数组下标从 1 算起！
建成的完美线段树如下图：

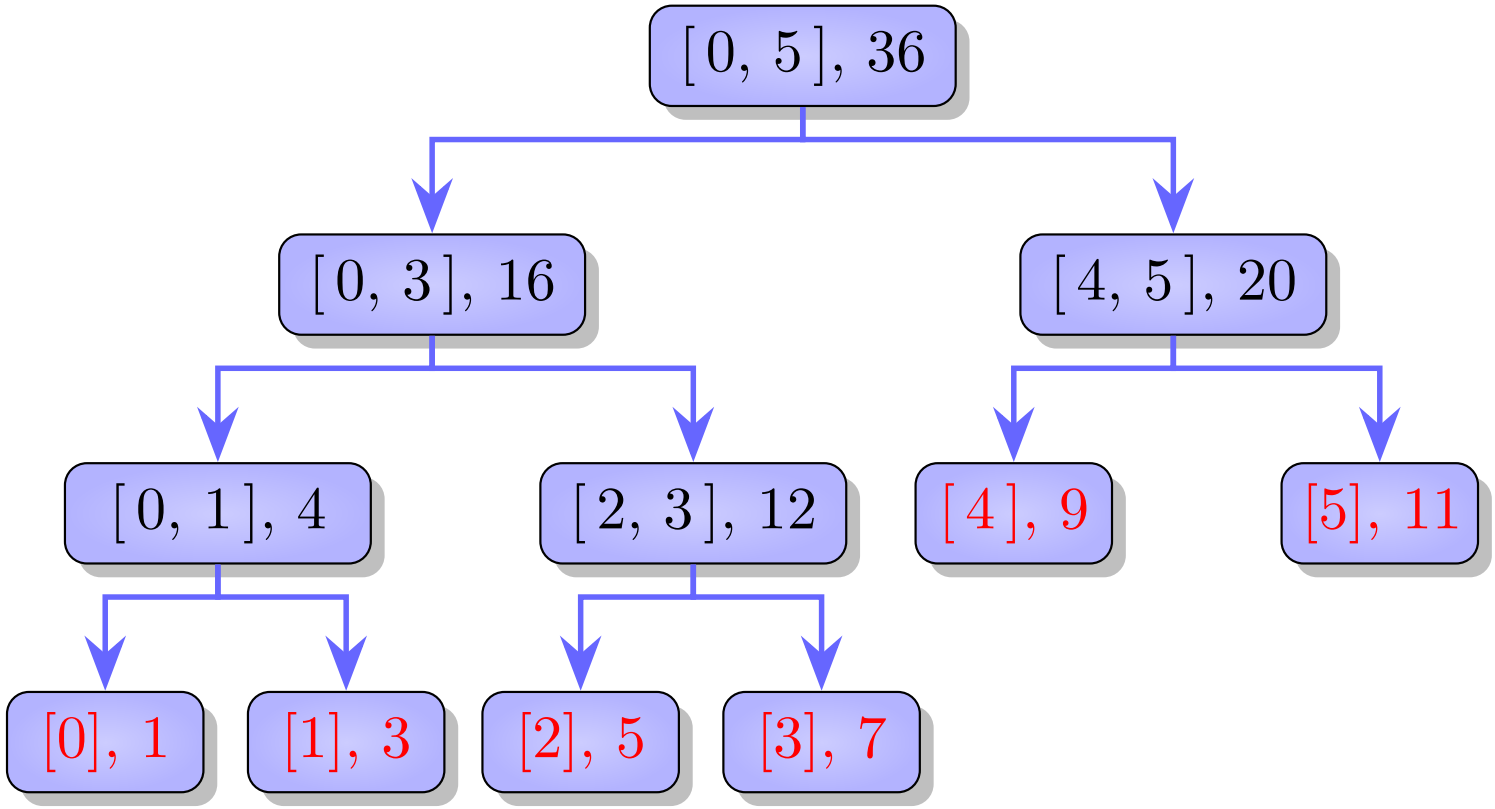


线段树数组表示为 $st = \{36, 16, 20, 4, 12, 9, 11, 1, 3, 7, 9\}$，
这棵树好看了，完美了！那么线段树数组大小是多少？

$2 \times n - 1$

我们的新梦想

数据数组 $arr = \{1, 3, 5, 7, 9, 11\}$，注意数组下标从 1 算起！
建成的完美线段树如下图：



线段树数组表示为 $st = \{36, 16, 20, 4, 12, 9, 11, 1, 3, 7, 9\}$，
这棵树好看了，完美了！那么线段树数组大小是多少？
$2 \times n - 1$

新梦想破碎

同学们,这个梦想中的线段树还是存在这很多不完美的地方:

1. $st = \{36, 16, 20, 4, 12, 9, 11, 1, 3, 7, 9\}$,原串在线段树数组中顺序是错的。这对我们些程序,计算和统计非常不方便。
2. 所有的原数据不再一个层中。同样不方便我们计算。

如果这棵二叉树是完全二叉树,甚至是满二叉树;所有的数据数据都按顺序排列在线段树的最下层叶节点。那就完美了。

新梦想破碎

同学们,这个梦想中的线段树还是存在这很多不完美的地方:

1. $st = \{36, 16, 20, 4, 12, 9, 11, 1, 3, 7, 9\}$,原串在线段树数组中顺序是错的。这对我们些程序,计算和统计非常不方便。
2. 所有的原数据不再一个层中。同样不方便我们计算。

如果这棵二叉树是完全二叉树,甚至是满二叉树;所有的数据数据都按顺序排列在线段树的最下层叶节点。那就完美了。

新梦想破碎

同学们,这个梦想中的线段树还是存在这很多不完美的地方:

1. $st = \{36, 16, 20, 4, 12, 9, 11, 1, 3, 7, 9\}$,原串在线段树数组中顺序是错的。这对我们些程序,计算和统计非常不方便。
2. 所有的原数据不再一个层中。同样不方便我们计算。

如果这棵二叉树是完全二叉树,甚至是满二叉树;所有的数据数据都按顺序排列在线段树的最下层叶节点。那就完美了。

新梦想破碎

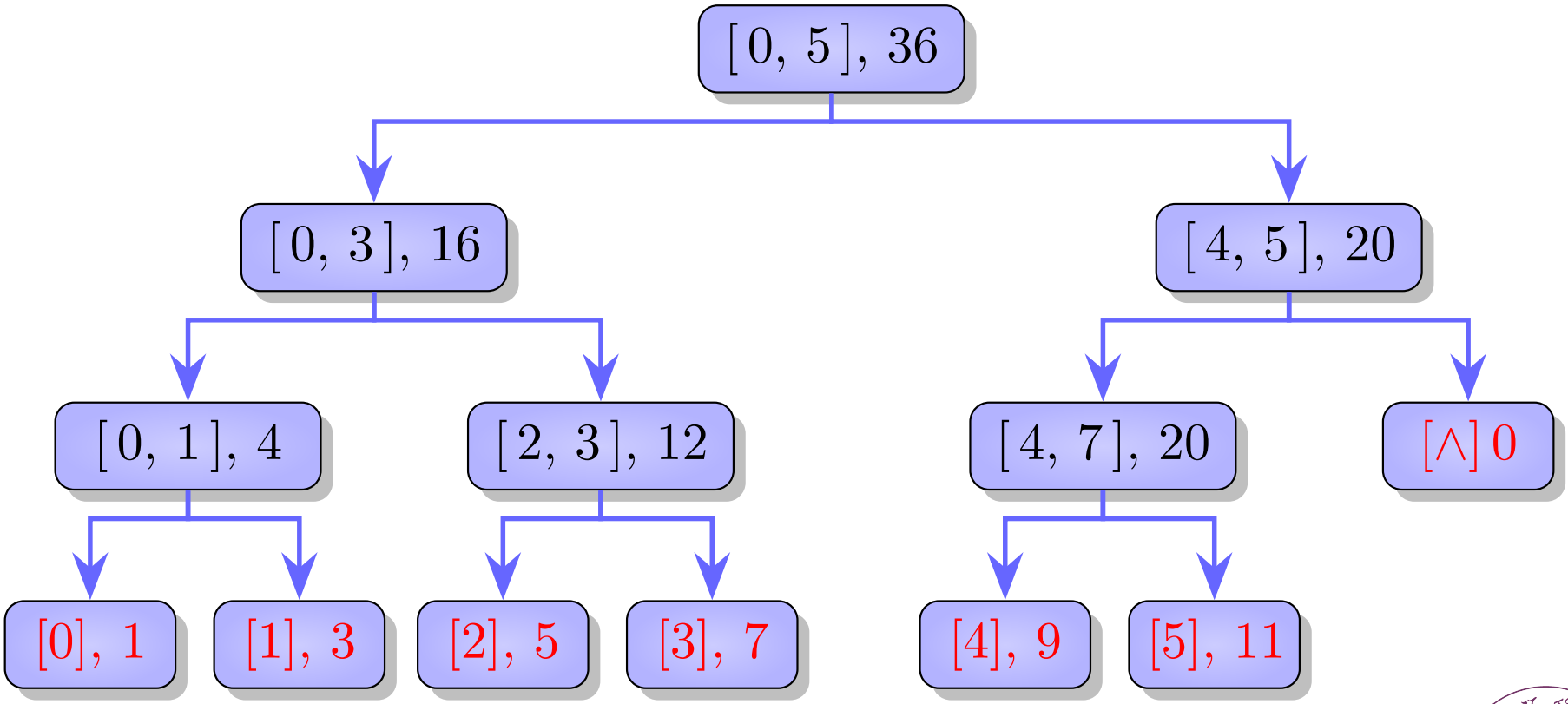
同学们,这个梦想中的线段树还是存在这很多不完美的地方:

1. $st = \{36, 16, 20, 4, 12, 9, 11, 1, 3, 7, 9\}$,原串在线段树数组中顺序是错的。这对我们些程序,计算和统计非常不方便。
2. 所有的原数据不再一个层中。同样不方便我们计算。

如果这棵二叉树是完全二叉树,甚至是满二叉树; 所有的数据数据都按顺序排列在线段树的最下层叶节点。那就完美了。

最终答案

数据数组 $arr = \{1, 3, 5, 7, 9, 11\}$，注意数组下标从 1 算起！
建成的完美线段树如下图：



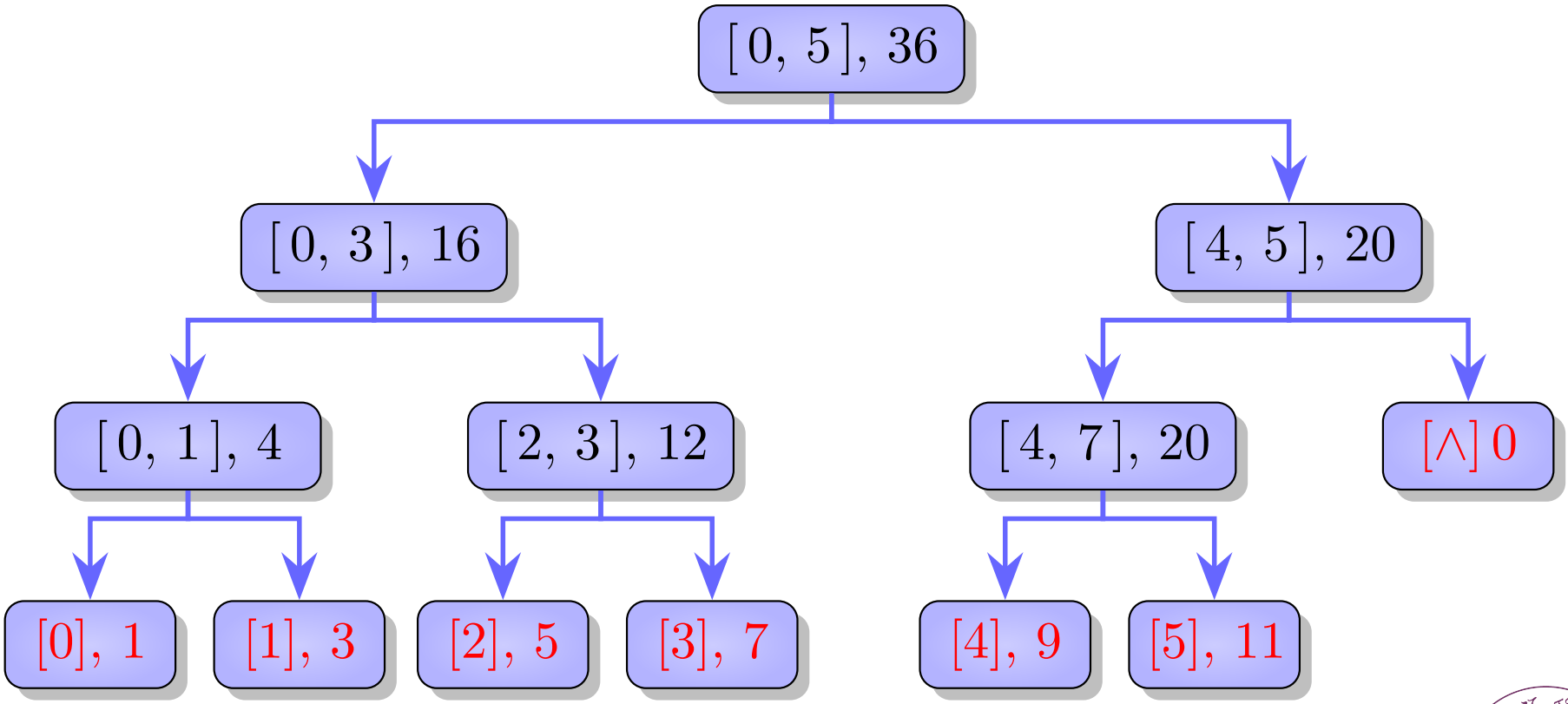
线段树数组表示为

$st = \{36, 16, 20, 4, 12, 9, 0, 1, 3, 5, 7, 9, 11\}$，

这棵树好看了，完美了！那么线段树数组大小是多少？自己算！

最终答案

数据数组 $arr = \{1, 3, 5, 7, 9, 11\}$,注意数组下标从 1 算起!
建成的完美线段树如下图:



线段树数组表示为
$st = \{36, 16, 20, 4, 12, 9, 0, 1, 3, 5, 7, 9, 11\}$,
这棵树好看了,完美了! 那么线段树数组大小是多少? 自己算!

非递归创建完美线段树 I

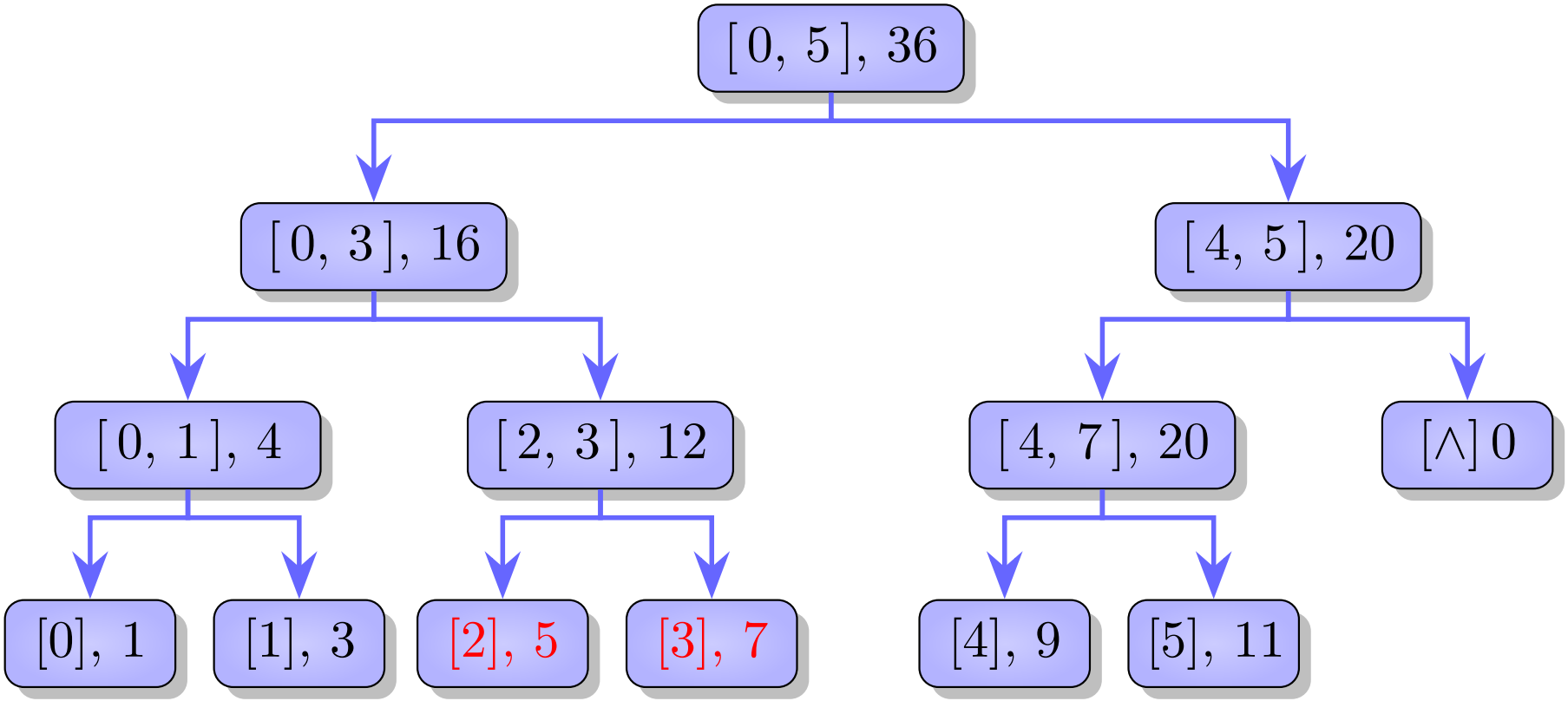
```
1  //两个参数分别为原数组和原数据个数
2  int *buildST(int arr[], int n){
3     int _2power=1;
4     while(_2power<n){
5        _2power=_2power<<1;
6     }
7     int N = _2power <<1;//计算线段树大小
8
9     int *st = new int[N];
10
11    memset(st,0,sizeof(int)*(_2power+n));
12
13    for(int i=_2power;i<_2power+n;++i){
14       st[i]=arr[i-_2power];
15    }
```

非递归创建完美线段树 I

```
17    for (int i=_2power-1;i>0;i--)
18      st[i]=st[2*i]+st[2*i+1];
19
20    //打印输出创建的线段树, 正式使用时删掉
21    for (int i=1;i<_2power+n;i++)
22      printf("T[%d] = %d\n",i,st[i]);
23
24    return st;
25  }
```

第一种情况

查询 $[2, 3]$!



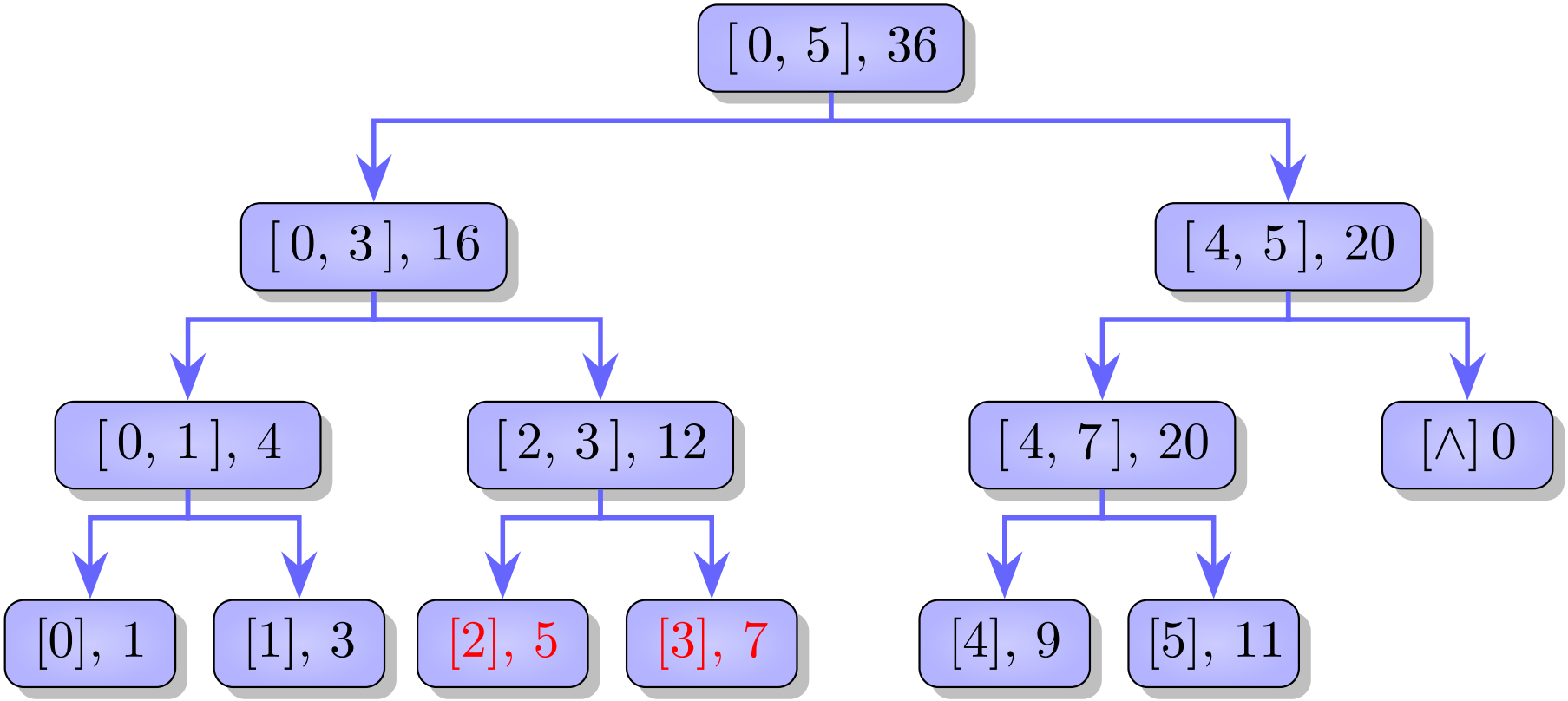
还采用自顶向下递归调用的方式找到节点?

因为线段树中完美的包含了原数组,可以采用非递归方法,通过如何建立线段树的方式计算下标 $(N >> 1) + i$.

所以 $[2, 3] = st[(N >> 1) + 2] + st[(N >> 1) + 2]$

第一种情况

查询 $[2, 3]$!



还采用自顶向下递归调用的方式找到节点?

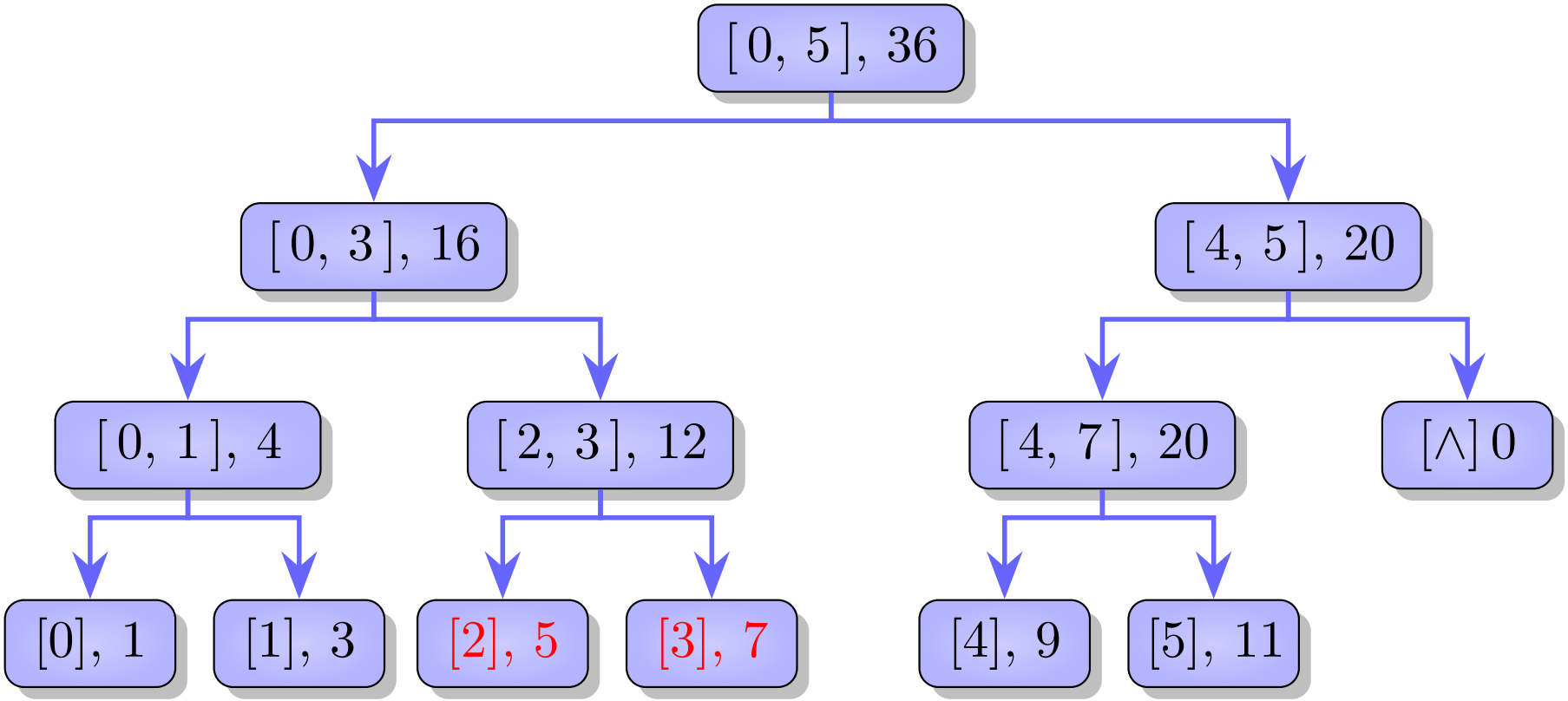
因为线段树中完美的包含了原数组,可以采用非递归方法,通过

建立线段树的方式计算下标 $(N >> 1) + i$.

所以 $[2, 3] = st[(N >> 1) + 2] + st[(N >> 1) + 2]$

第一种情况

查询 $[2, 3]$!



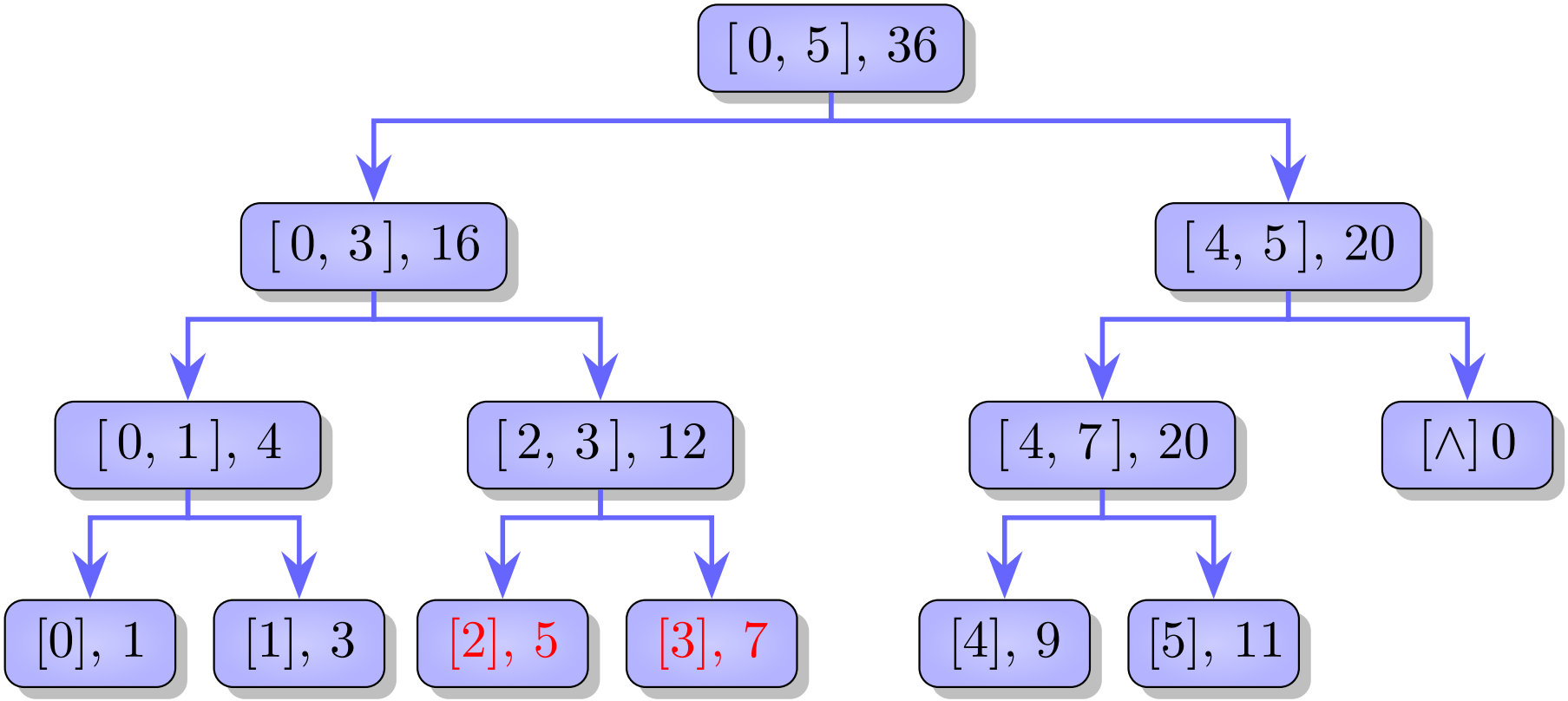
还采用自顶向下递归调用的方式找到节点?

因为线段树中完美的包含了原数组,可以采用非递归方法,通过前面建立线段树的方式计算下标 $(N >> 1) + i$.

所以 $[2, 3] = st[(N >> 1) + 2] + st[(N >> 1) + 2]$

第一种情况

查询 $[2, 3]$!



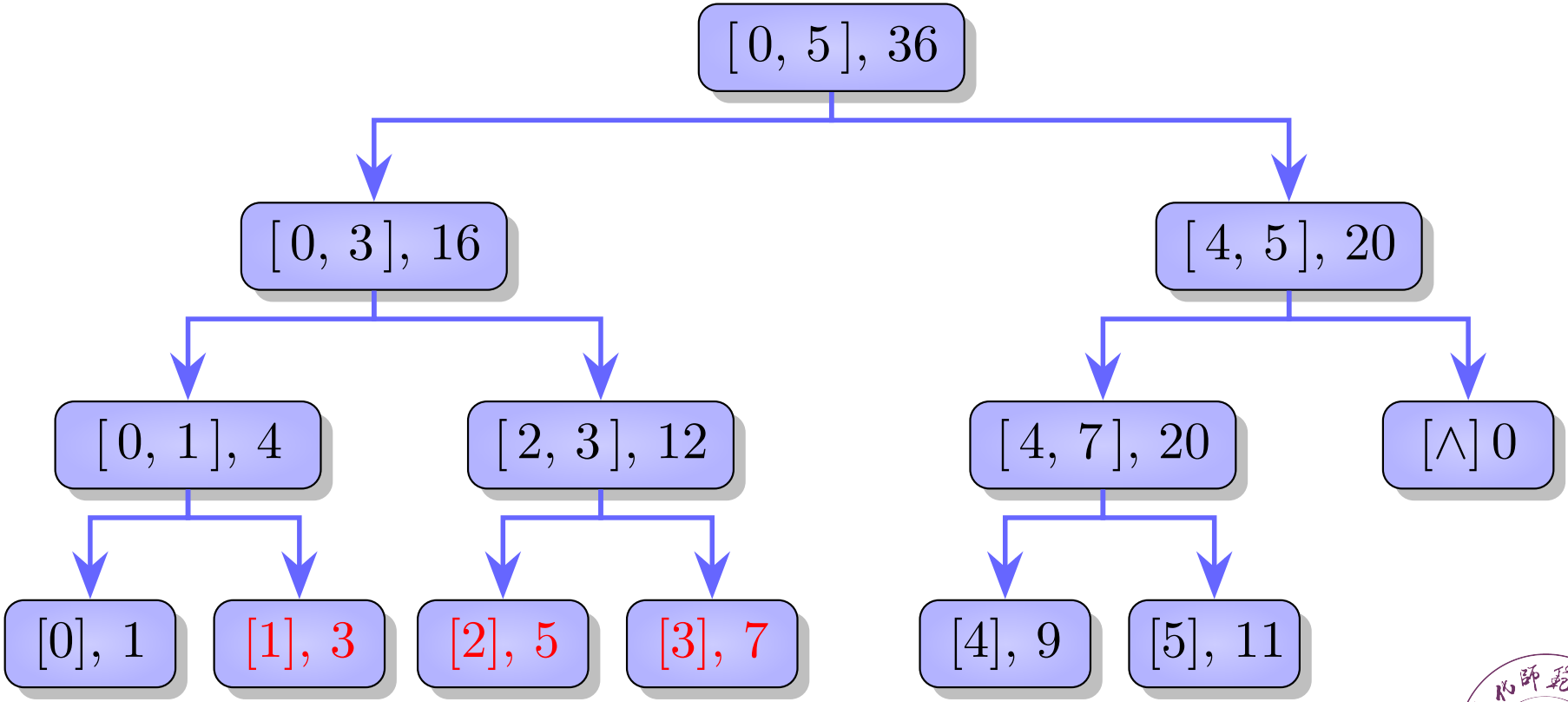
还采用自顶向下递归调用的方式找到节点?

因为线段树中完美的包含了原数组,可以采用非递归方法,通过前面建立线段树的方式计算下标 $(N >> 1) + i$.

所以 $[2, 3] = st[(N >> 1) + 2] + st[(N >> 1) + 2]$.

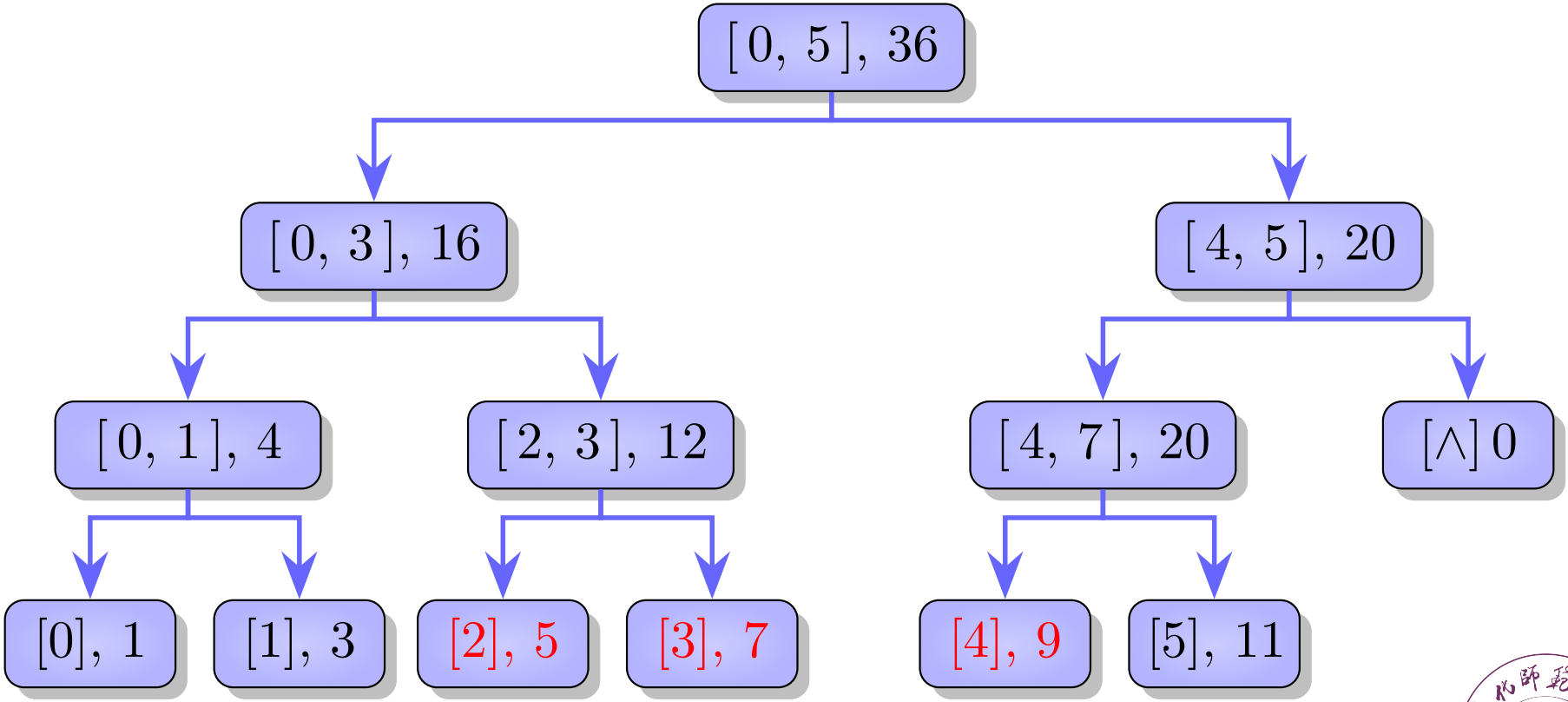
第二种情况

查询 $[1, 3]$!



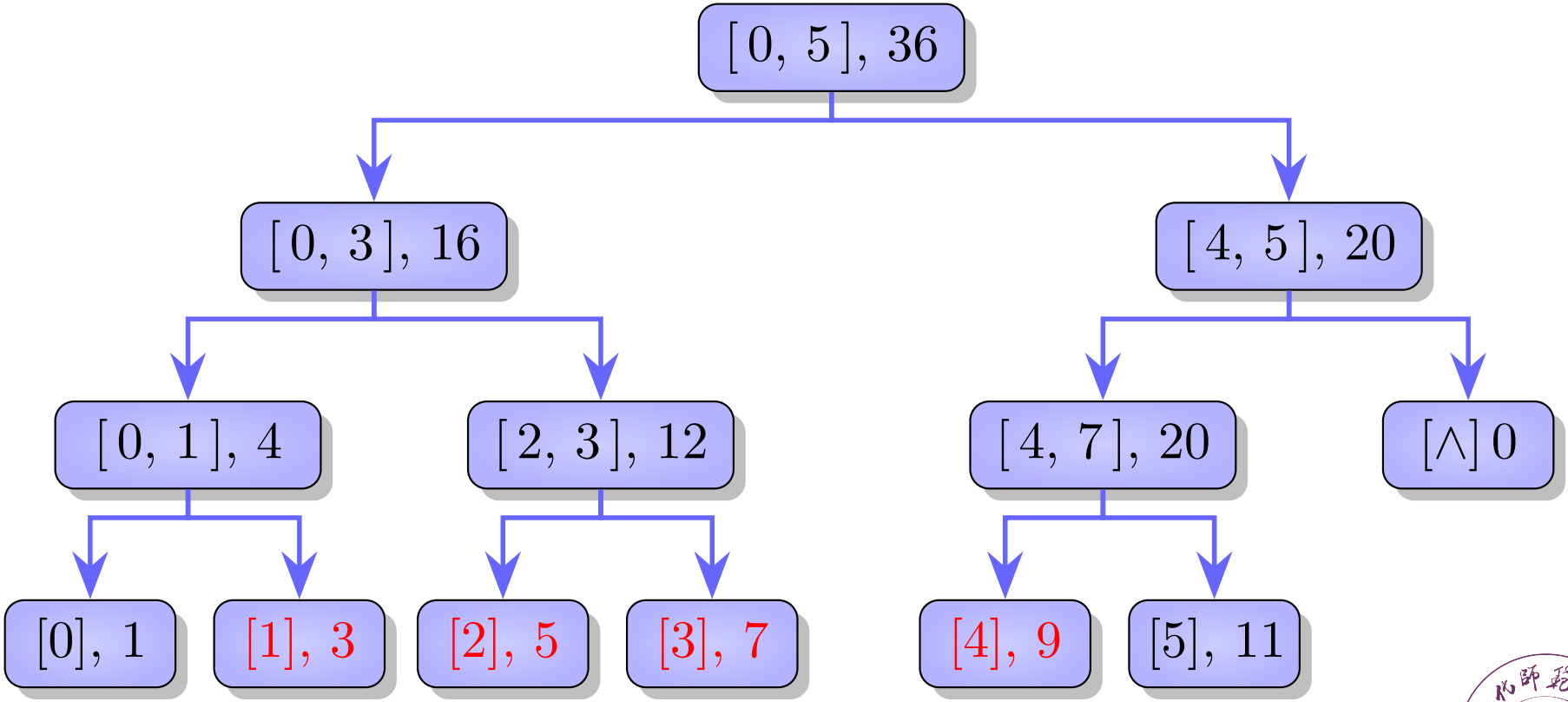
第三种情况

查询 $[2, 4]$!



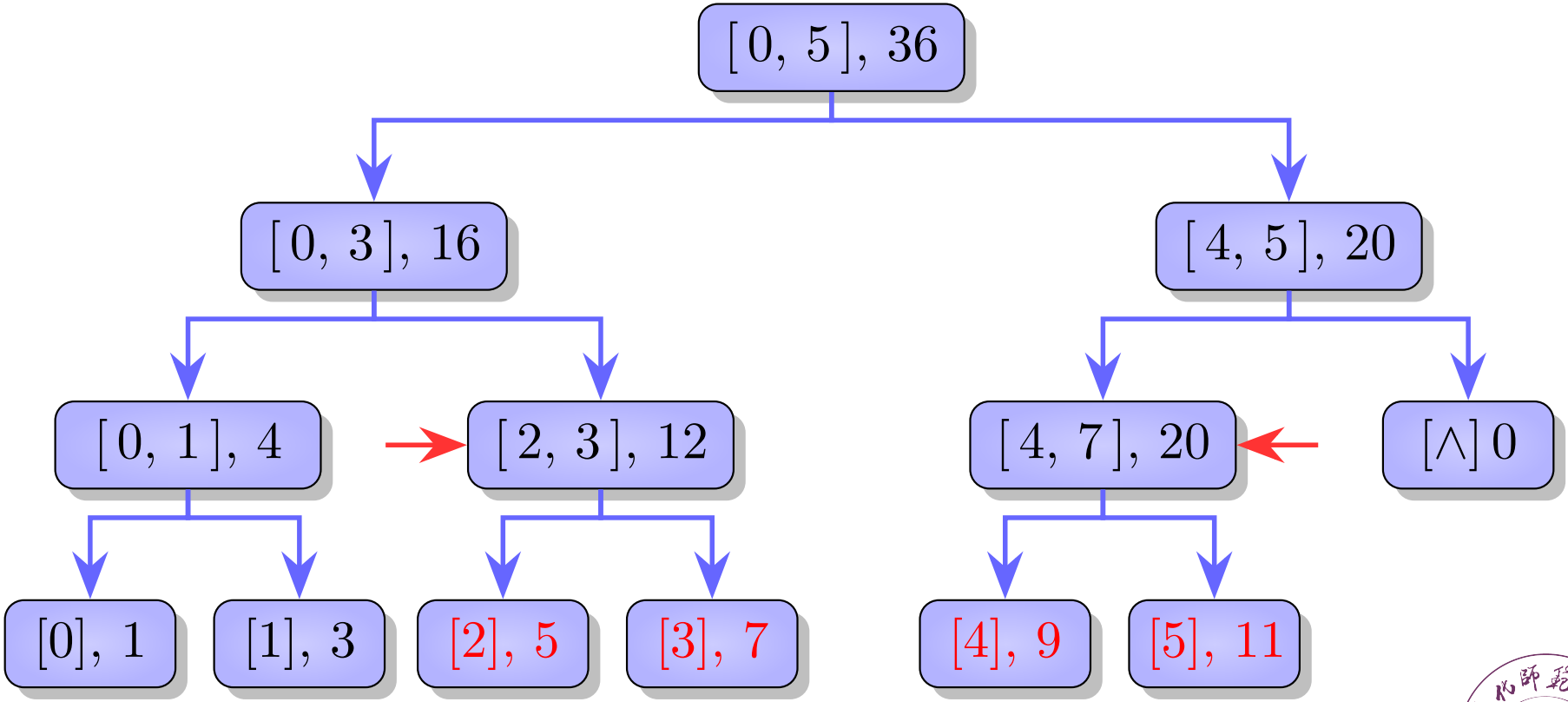
第四种情况

查询 $[1, 4]$!



第五种情况

查询 $[2, 5]$!



发现

这里使用的还是同学们高中学到的知识,分类!

1、起点和终点差为 1:直接相加!并返回结果!

2、起点是左儿子,终点是右儿子:起点终点都向上一层!

3、起点是右儿子:加起点,起点后移一位!

4、终点是左儿子:加终点,终点前移一位!

发现

这里使用的还是同学们高中学到的知识,分类!

1、起点和终点差为 1:直接相加! 并返回结果!

2、起点是左儿子,终点是右儿子:起点终点都向上一层!

3、起点是右儿子:加起点,起点后移一位!

4、终点是左儿子:加终点,终点前移一位!

发现

这里使用的还是同学们高中学到的知识,分类!

1、起点和终点差为 1:直接相加! 并返回结果!

2、起点是左儿子,终点是右儿子:起点终点都向上一层!

3、起点是右儿子:加起点,起点后移一位!

4、终点是左儿子:加终点,终点前移一位!

发现

这里使用的还是同学们高中学到的知识,分类!
1、起点和终点差为 1:直接相加! 并返回结果!
2、起点是左儿子,终点是右儿子:起点终点都向上一层!
3、起点是右儿子:加起点,起点后移一位!
4、终点是左儿子:加终点,终点前移一位!

发现

这里使用的还是同学们高中学到的知识,分类!
1、起点和终点差为 1:直接相加! 并返回结果!
2、起点是左儿子,终点是右儿子:起点终点都向上一层!
3、起点是右儿子:加起点,起点后移一位!
4、终点是左儿子:加终点,终点前移一位!

非递归查询完美线段树

```
1  int getSum(int *st, int n, int qs, int qe){
2    int sum=0;
3    int _2power=1;
4    while(_2power<n){ _2power=_2power<<1; }
5    int s=_2power+qs-1;
6    int e=_2power+qe-1;
7    while(e>s){
8      if(e-s==1) return sum+st[e]+st[s];
9      if(s%2==1 && e%2==0) s/=2,e/=2;
10     if(s%2==0) sum+=st[s], s+=1;
11     if(e%2==1) sum+=st[e], e-=1;
12   }
13   return sum;
14 }
```

加油!!!

同学们课后要努力!!!!
自己思考实现更新代码!!!!
将讲义中代码组合并运行!!!!