区间问题处理技巧

- 前缀和
- 差分数组
- 稀疏表
- 尺取法
- 线段树简介
- 树状数组

数据之法
结构之美
算法之道

Last updated on 2024.11

zhuyungang@jlu.edu.cn

区间问题处理技巧

- **前缀和**
- 差分数组
- 稀疏表
- 尺取法
- 线段树简介
- 树状数组

数据之法
结构之美
算法之道

Last updated on 2024.11

zhuyungang@jlu.edu.cn

区间求和问题

本质：给定一个长度为*n*的数组*a*，做*m*次查询，每次查询区间*i*至*j*的元素之和。【洛谷B3612】

a						
			i	……	*j*	

暴力方法：

```
while(m--){
    scanf("%d %d",&i,&j);
    int sum=0;
    for(int k=i; k<=j; k++)
        sum+=a[k];
}
```

区间求和问题

本质：给定一个长度为*n*的数组*a*，做*m*次查询，每次查询区间*i*至*j*的元素之和。【洛谷B3612】

a

i …… *j*

暴力方法：

```
while(m--){
    scanf("%d %d",&i,&j);
    int sum=0;
    for(int k=i; k<=j; k++)
        sum+=a[k];
}
```

时间复杂度 O(*nm*)

前缀和：一种重要的数据预处理技巧

- 令 $sum_i = a_1 + a_2 + \ldots + a_i$，
- 则 $a_i + \ldots + a_j = sum_j - sum_{i-1}$

每次区间求和O(1)

$$\underbrace{\overbrace{\boxed{a_1}\boxed{\;\ldots\;}\boxed{a_{i-1}}}^{sum_{i-1}}\boxed{a_i}\boxed{\;\ldots\;}\boxed{a_j}}_{sum_j}$$

前缀和：一种重要的数据预处理技巧

- 令$sum_i = a_1+a_2+\ldots+a_i$，
- $sum_i = a_1+a_2+\ldots+a_{i-1}+a_i$
  $= sum_{i-1}+a_i$

```
sum[0]=0;
for(int i=1; i<=n; i++)
    sum[i]=sum[i-1]+a[i];
```

【洛谷B3612】

```
#include<stdio.h>
using namespace std;
const int N = 1e5 + 10;
int main() {
	int sum[N], a[N], n, m, i, j;
	sum[0] = 0;
	scanf("%d", &n);
	for (i = 1; i <= n; i++) {
		scanf("%d", &a[i]);
		sum[i] = sum[i-1] + a[i];
	}
	scanf("%d", &m);
	while (m--) {
		scanf("%d %d", &i, &j);
		printf("%d\n", sum[j]-sum[i-1]);
	}
	return 0;
}
```

$O(n+m)$

前缀和技巧适用场合：数组在不被修改的情况下，频繁查询某个区间的累加和。

预处理$O(n)$
每次区间查询$O(1)$

区间问题处理技巧

- 前缀和
- **差分数组**
- 稀疏表
- 尺取法
- 线段树简介
- 树状数组

数据之法

结构之美

算法之道

Last updated on 2024.11

zhuyungang@jlu.edu.cn

差分数组：一种重要的数据预处理技巧

给定一个长度为*n*的数组*a*（下标从1开始），做m次操作，每次操作为3个整数*i*、*j*、*d*，表示对区间*i*至*j*的所有元素加上*d*，返回最后的数组。【洛谷P2367、LeetCode1109】

示例：

操作		1	2	3	4	5	6	7	8	9
	a	6	10	20	29	35	50	60	70	80
3 6 8		6	10	**28**	**37**	**43**	**58**	60	70	80
4 7 10		6	10	28	**47**	**53**	**68**	**70**	70	80
2 8 2		6	**12**	**30**	**49**	**55**	**70**	**72**	**72**	80

给定一个长度为n的数组a，做m次操作，每次操作为3个整数i、j、d，表示对区间i至j的所有元素加上d。

暴力方法：

```
while(m--){
   scanf(“%d %d %d”,&i,&j,&d);
   for(int k=i; k<=j; k++)
      a[k]+=d;
}
```

给定一个长度为*n*的数组*a*，做*m*次操作，每次操作为3个整数*i*、*j*、*d*，表示对区间*i*至*j*的所有元素加上*d*。

暴力方法：

```
while(m--){
   scanf("%d %d %d",&i,&j,&d);
   for(int k=i; k<=j; k++)
      a[k]+=d;
}
```

时间复杂度 $O(nm)$

差分数组 $diff[i]=a[i]-a[i-1]$

	1	2	3	4	5	6	7	8
a	1	2	5	6	9	8	10	7
$diff$								

差分数组 $diff[i]=a[i]-a[i-1]$

	1	2	3	4	5	6	7	8
a	1	2	5	6	9	8	10	7
$diff$	1	1	3	1	3	-1	2	-3

差分数组 $diff[i]=a[i]-a[i-1]$

	1	2	3	4	5	6	7	8
a	1	2	5+**d**	6+**d**	9+**d**	8+**d**	10	7
$diff$	1	1	3+**d**	1	3	-1	2-**d**	-3

i j

```
diff[i]+=d;
if(j<n) diff[j+1]-=d;
```

1次区间操作
O(1)

给定一个长度为*n*的数组*a*（下标从1开始），做*m*次操作，每次操作是对区间*i*至*j*的所有元素加上*d*，返回最后的数组。

```
const int N=5e6+10;
void RangeIncrement(int a[],int n,int m){
      int diff[N],i,j,d;
      diff[1]=a[1];    //计算差分数组
      for(int i=2;i<=n;i++)
            diff[i]=a[i]-a[i-1];
      while(m--){         //a[i]…a[j]加d
            scanf("%d %d %d",&i,&j,&d);
            diff[i]+=d;
            if(j<n) diff[j+1]-=d;
      }
      a[1]=diff[1];    //利用diff反推/恢复数组a
      for(int i=2; i<=n; i++)
            a[i]=a[i-1]+diff[i];
}
```

$diff[i]=a[i]-a[i-1]$

$O(n+m)$

预处理$O(n)$
每次区间操作$O(1)$

差分数组适用场合：频繁对数组的某个区间的元素增减固定值。

区间问题处理技巧

- 前缀和
- 差分数组
- **稀疏表（Sparse Table）**
- 尺取法
- 线段树简介
- 树状数组

数据之法
结构之美
算法之道

zhuyungang@jlu.edu.cn

区间最值问题（*Range Maximum/Minimum Query, RMQ*）

给定一个长度为*n*的数组*a*（下标从1开始），做*m*次查询，每次查询为2个整数*i*、*j*，表示区间*i*至*j*的最大值。【洛谷P3865】

示例：

	1	2	3	4	5	6	7	8	9
a	3	9	1	2	5	6	0	7	8

查询	输出
3 6	6
5 8	7
2 7	9

区间最值问题（*Range Maximum/Minimum Query, RMQ*）

给定一个长度为*n*的数组*a*（下标从1开始），做*m*次查询，每次查询为2个整数*i*、*j*，表示区间*i*至*j*的最大值。

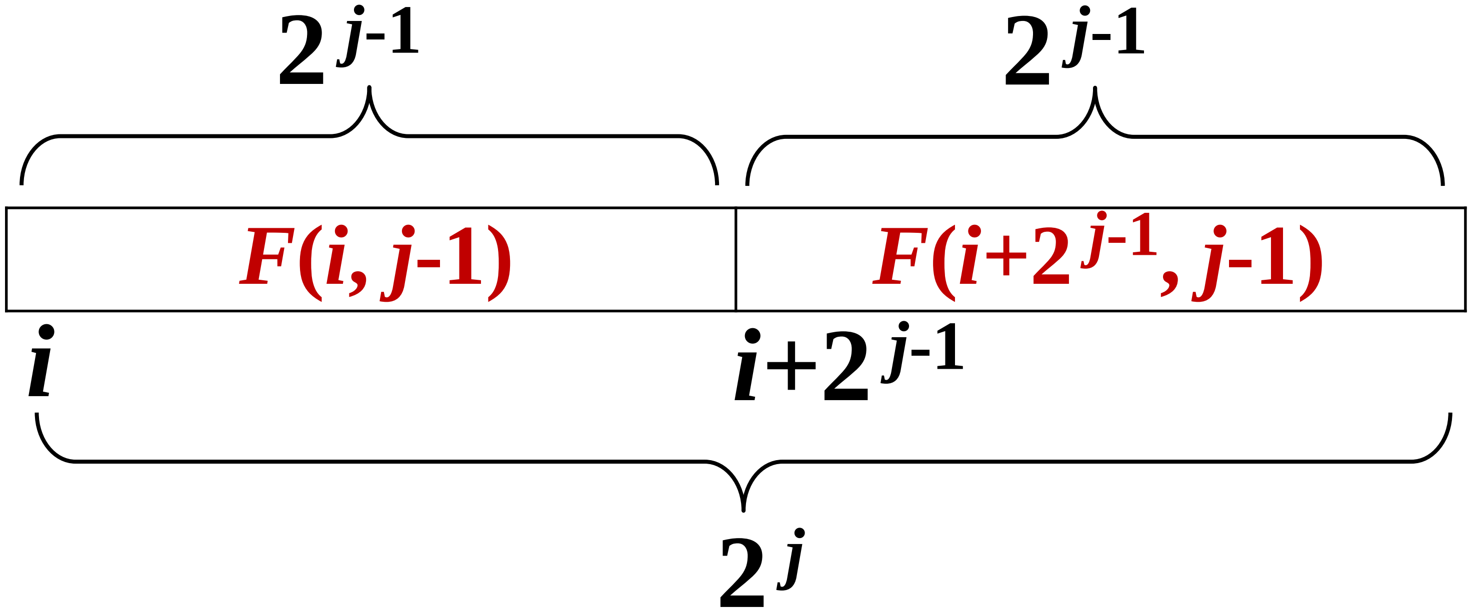
暴力方法


```
while(m--){
    scanf("%d %d",&i,&j);
    int max=INT_MIN;
    for(int k=i;k<=j;k++)
        if(a[k]>max) max=a[k];
}
```

时间复杂度 $O(nm)$

稀疏表（Sparse Table）

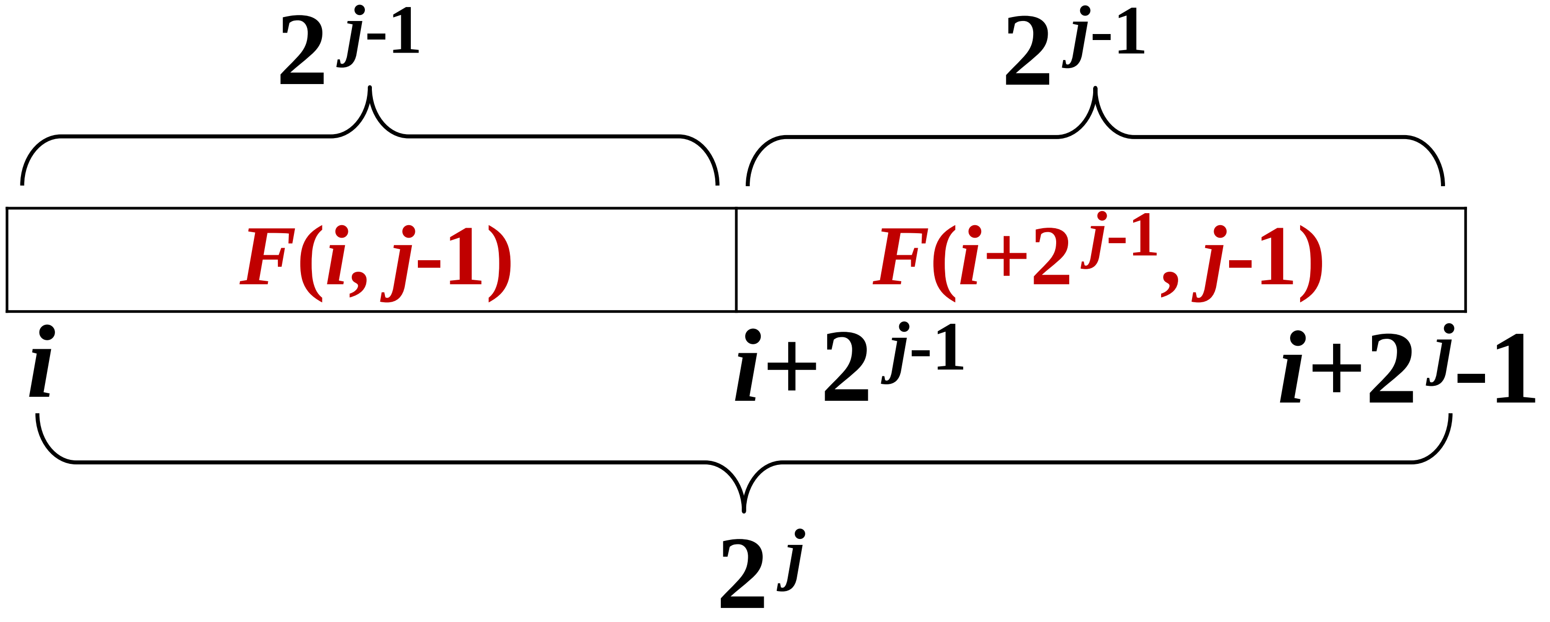
- 令$F(i, j)$表示数组A中从下标i开始的2^j个数的最大值，即子区间$[i, i+2^j-1]$的最大值。
- 区间$[i, i+2^j-1]$由两个区间组成：①下标i开始的长度为2^{j-1}的区间，其最值为$F(i, j-1)$；②下标$i+2^{j-1}$开始的长度为2^{j-1}的区间，其最值为$F(i+2^{j-1}, j-1)$；



稀疏表（Sparse Table）

- 长度为2^j的区间的最大值是左右两半长度为2^{j-1}的区间的最大值中的较大者。

$$F(i,j)=\begin{cases}\max\{F(i,j-1),F(i+2^{j-1},j-1)\}, & j>0\\ a[i], & j=0\end{cases}$$



$0 \le j \le \log_2 n$

$1 \le i \le n\text{-}2^j\text{+}1$

稀疏表（Sparse Table）

$$F(i,j)=\begin{cases}\max\{F(i,j-1),F(i+2^{j-1},j-1)\}, & j>0\\ a[i], & j=0\end{cases}$$

		***j*-1**	***j***	
i		***F*[*i*][*j*-1]**	***F*[*i*][*j*]**	
		⋮		
***i*+2** $i+2^{j-1}$		$F[i+2^{j-1}][j-1]$		

如何计算2^j

十进制表示	二进制表示	相当于
2^0	0 0 0 1	

如何计算2^j

十进制表示	二进制表示	相当于
2^0	0 0 0 1	
2^1	0 0 1 0	1左移1位

如何计算2^j

十进制表示	二进制表示	相当于
2^0	0 0 0 1	
2^1	0 0 1 0	1左移1位
2^2	0 1 0 0	1左移2位

如何计算2^j

十进制表示	二进制表示	相当于
2^0	0 0 0 1	
2^1	0 0 1 0	1左移1位
2^2	0 1 0 0	1左移2位
2^3	1 0 0 0	1左移3位

2^j：将1对应的二进制数左移j位

C/C++位运算： 1<<j

构建ST表 洛谷P3865

```c
int main() {
    int n,m,L,R,a[maxn],F[maxn][maxlogn];
    scanf("%d %d", &n, &m);
    for (int i=1; i<=n; i++) {
        scanf("%d", &a[i]);
        F[i][0] = a[i];
    }
    int logn = log2(n);
    for (int j=1; j<=logn; j++)
        for (int i=1; i<=n-(1<<j)+1; i++)
            F[i][j]=max(F[i][j-1], F[i+(1<<(j-1))][j-1]);

    return 0;
}
```

		j-1	j	
i		$F[i][j\text{-}1]$	$F[i][j]$	
		⋮		
$i+2^{j-1}$		$F[i+2^{j-1}][j\text{-}1]$		

$0 \le j \le \log_2 n$

$1 \le i \le n\text{-}2^j+1$

$$F(i,j)=\begin{cases}\max\{F(i,j-1),F(i+2^{j-1},j-1)\}, & j>0\\ a[i], & j=0\end{cases}$$

时间复杂度

O(*n*log*n*)

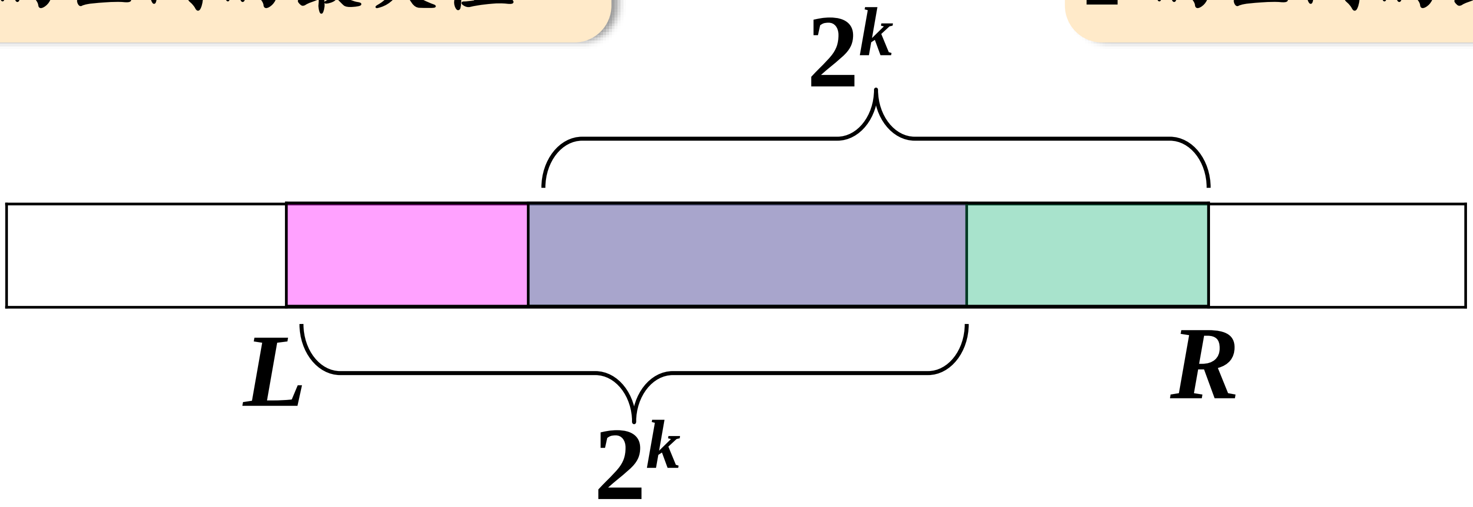
```c
const int maxn=1e5+10;
const int maxlogn=18;
int max(int a, int b){
    return (a>b)?a:b;
}
```

查询区间[L, R]的最值

- 令$F(i, j)$表示数组A中以下标i为起点的长度为2^j的区间的最大值，即子区间$[i, i+2^j-1]$的最大值。
- 令$k=\lfloor \log_2(R-L+1) \rfloor$
- 区间[L, R]的最大值= $max\{ F[L][k]$ ，$F[R-2^k+1][k] \}$

以L为起点、长度为2^k的区间的最大值

以R为终点、长度为2^k的区间的最大值



每次查询
O(1)

ST表 洛谷P3865

```
int main() {
    int n,m,L,R,a[maxn],F[maxn][maxlogn];
    scanf("%d %d", &n, &m);
    for (int i=1; i<=n; i++) {
        scanf("%d", &a[i]);
        F[i][0] = a[i];
    }
    int logn = log2(n);
    for (int j=1; j<=logn; j++)
        for (int i=1; i<=n-(1<<j)+1; i++)
            F[i][j]=max(F[i][j-1], F[i+(1<<(j-1))][j-1]);
    while(m--){
        scanf("%d %d", &L, &R);
        int k = log2(R - L + 1);
        int ans=max(F[L][k],F[R-(1<<k)+1][k]);
        printf("%d\n",ans);
    }
    return 0;
}
```

时间复杂度O(nlogn+m)

预处理O(nlogn)
每次区间查询O(1)

倍增思想

```
const int maxn=1e5+10;
const int maxlogn=18;
int max(int a, int b){
    return (a>b)?a:b;
}
```

$max\{ F[L][k],\ F[R\text{-}2^k\text{+}1][k] \}$

区间问题处理技巧

- 前缀和
- 差分数组
- 稀疏表
- **尺取法**
- 线段树简介
- 树状数组

数据之法
结构之美
算法之道

Last updated on 2024.11

zhuyungang@jlu.edu.cn

例：给定一个长度为n的正整数数组a和一个整数S，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。

示例：

数组a： 6 1 2 3 4 6 4 1 1 8 9，S=6

输出：

0 0

1 3

5 5

6 8

例：给定一个长度为n的**正整数**数组a和一个整数S，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。

暴力方案：

```
for(int i=0;i<n;i++)
    for(int j=i;j<n;j++){
        //看a[i]+…+a[j]是否等于S
    }
```

例：给定一个长度为*n*的正整数数组*a*和一个整数*S*，在这个数组中找出元素之和等于*S*的所有区间，输出区间的起点和终点位置。**以S=9为例**

5	2	3	6	2	7	1	5	3	2

例：给定一个长度为*n*的正整数数组*a*和一个整数*S*，在这个数组中找出元素之和等于*S*的所有区间，输出区间的起点和终点位置。以S=9为例

5	2	3	6	2	7	1	5	3	2

- 区间和 < S，区间右边扩1位（区间终点推进1位）
- 区间和 ≥ S，区间左边缩1位（区间起点推进1位）

例：给定一个长度为*n*的正整数数组*a*和一个整数*S*，在这个数组中找出元素之和等于*S*的所有区间，输出区间的起点和终点位置。**以S=9为例**

5	**2**	3	6	2	7	1	5	3	2

例：给定一个长度为*n*的正整数数组*a*和一个整数*S*，在这个数组中找出元素之和等于*S*的所有区间，输出区间的起点和终点位置。以S=9为例

5	2	3	6	2	7	1	5	3	2

例：给定一个长度为*n*的正整数数组*a*和一个整数*S*，在这个数组中找出元素之和等于*S*的所有区间，输出区间的起点和终点位置。以S=9为例

5	**2**	**3**	6	2	7	1	5	3	2

例：给定一个长度为n的正整数数组a和一个整数S，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。以$S=9$为例

5	**2**	**3**	**6**	2	7	1	5	3	2

例：给定一个长度为n的正整数数组a和一个整数S，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。**以S=9为例**

5	2	**3**	**6**	2	7	1	5	3	2

找到

例：给定一个长度为n的正整数数组a和一个整数S，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。**以S=9为例**

5	2	3	**6**	2	7	1	5	3	2

例：给定一个长度为*n*的正整数数组*a*和一个整数*S*，在这个数组中找出元素之和等于*S*的所有区间，输出区间的起点和终点位置。以S=9为例

5	2	3	**6**	**2**	7	1	5	3	2

例：给定一个长度为n的正整数数组a和一个整数S，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。以S=9为例

5	2	3	**6**	**2**	**7**	1	5	3	2

例：给定一个长度为n的正整数数组a和一个整数S，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。以S=9为例

5	2	3	6	**2**	**7**	1	5	3	2

找到

例：给定一个长度为*n*的正整数数组*a*和一个整数*S*，在这个数组中找出元素之和等于*S*的所有区间，输出区间的起点和终点位置。**以S=9为例**

5	2	3	6	2	**7**	1	5	3	2

例：给定一个长度为n的正整数数组a和一个整数S，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。以S=9为例

5	2	3	6	2	**7**	**1**	5	3	2

例：给定一个长度为*n*的正整数数组*a*和一个整数*S*，在这个数组中找出元素之和等于*S*的所有区间，输出区间的起点和终点位置。以S=9为例

5	2	3	6	2	**7**	**1**	**5**	3	2

例：给定一个长度为*n*的正整数数组*a*和一个整数*S*，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。**以S=9为例**

5	2	3	6	2	7	**1**	**5**	3	2

例：给定一个长度为n的正整数数组a和一个整数S，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。**以S=9为例**

5	2	3	6	2	7	**1**	**5**	**3**	2

找到

例：给定一个长度为*n*的正整数数组*a*和一个整数*S*，在这个数组中找出元素之和等于*S*的所有区间，输出区间的起点和终点位置。**以S=9为例**

5	2	3	6	2	7	1	**5**	**3**	2

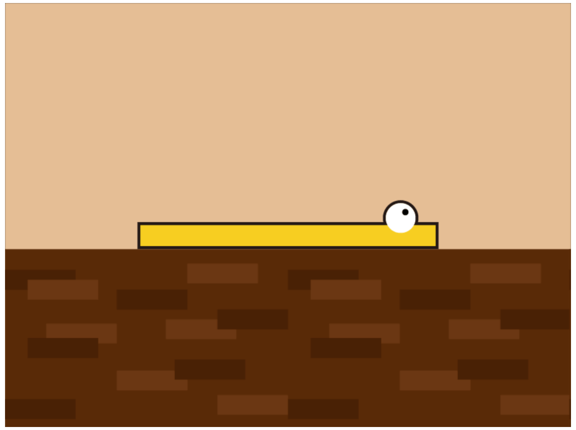
例：给定一个长度为n的正整数数组a和一个整数S，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。以S=9为例

5	2	3	6	2	7	1	**5**	**3**	**2**

例：给定一个长度为n的正整数数组a和一个整数S，在这个数组中找出元素之和等于S的所有区间，输出区间的起点和终点位置。**以S=9为例**

时间复杂度O(n)

5	2	3	6	2	7	1	5	**3**	**2**



尺取法：维护两个指针（下标）指向区间的起点和终点，根据实际情况交替推进两个指针（区间的左右边界），直到得出答案。

应用举例

给定一个长度为n的正整数数组a和一个正整数S，在这个数组中找出元素之和大于等于S的最短区间，返回该区间长度。若不存在满足条件的区间，返回0。【LeetCode209、POJ3061】

示例：

输入：a=[2 3 1 1 4 3]，S=6

输出：2

如果找到sum ≥ S的区间，则需要做两件事

① 找到了满足条件的区间，与当前最短区间比较

② 区间左边缩一位

```
int minSubArrayLen(int S, int a[], int n){
    int left=0,right=0,sum=a[0],min=n+1;
    while(true){
        while(right<n-1 && sum<S)
            sum+=a[++right];     //右扩一位
        if(sum<S) break;         //扩到头时区间和小于S
        int len=right-left+1;    //找到了一个和≥S的区间
        if(len<min) min=len;
        sum-=a[left++];          //左缩一位
    }
    if(min==n+1)
        min=0;
    return min;
}
```

如果找到sum ≥ S的区间，则需要做两件事

① 找到了满足条件的区间，与当前最短区间比较

② 区间左边缩一位

课下思考：若数组a中有负数，上述算法还行么？

区间问题处理技巧

- 前缀和
- 差分数组
- ST表
- 尺取法
- **线段树入门**
- 树状数组入门

数据之法 结构之美 算法之道

Last updated on 2024.11

zhuyungang@jlu.edu.cn

给你一个数组 nums，请你完成以下两类查询：

（1）更新数组 nums 下标对应的值。

（2）返回数组 nums 中索引 left 和索引 right 之间（包含）的元素的和，其中 left <= right。

【腾讯、字节跳动、微软、谷歌、亚马逊、Facebook、Twitter面试题】

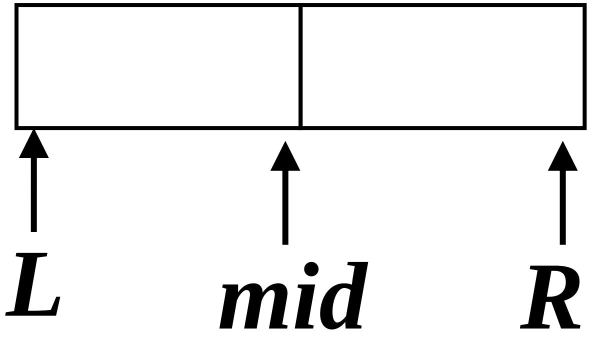
单点更新、区间求和问题

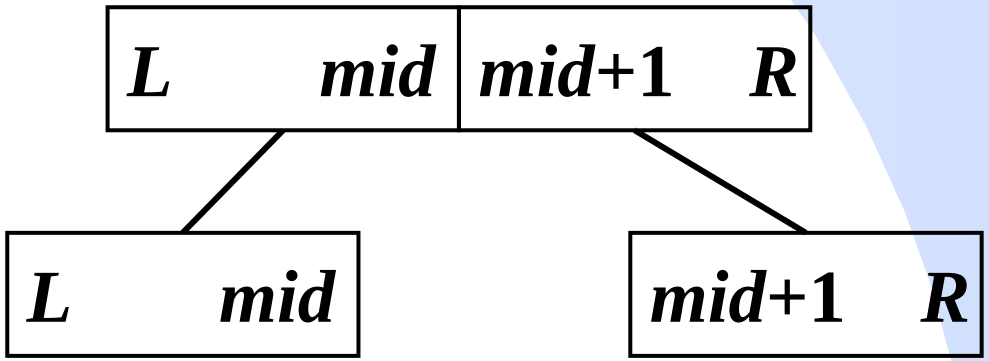
请编写程序对数组$a_1, a_2, \ldots, a_n$进行如下操作：

- ✓1 i x：给定i, x，将a_i加上x；
- ✓2 l r：给定l, r，求$a_l+a_{l+1}+\ldots+a_r$的值。

线段树（Segment Tree）

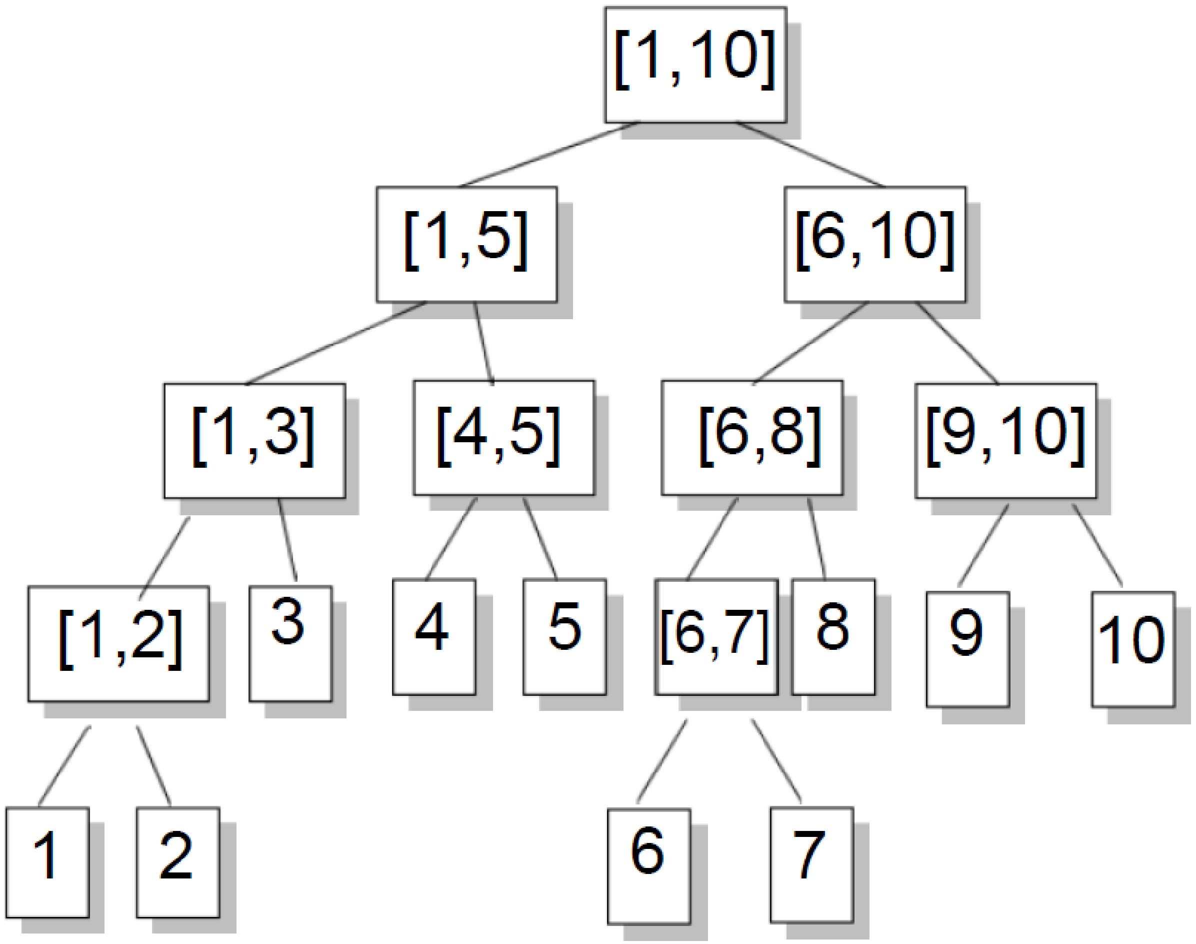
- 一棵二叉树，每个结点对应一个区间[*L*, *R*]。
- 根结点代表整个统计范围[1,*n*]。
- 每个叶结点代表一个长度为1的区间[*x*, *x*]。
- 对于每个非叶结点所表示的区间[*L*, *R*]，其左孩子表示的区间为[*L*, *mid*]，右孩子表示的区间为[*mid*+1, *R*]，其中*mid* = (*L*+*R*)/2。





线段树（Segment Tree）

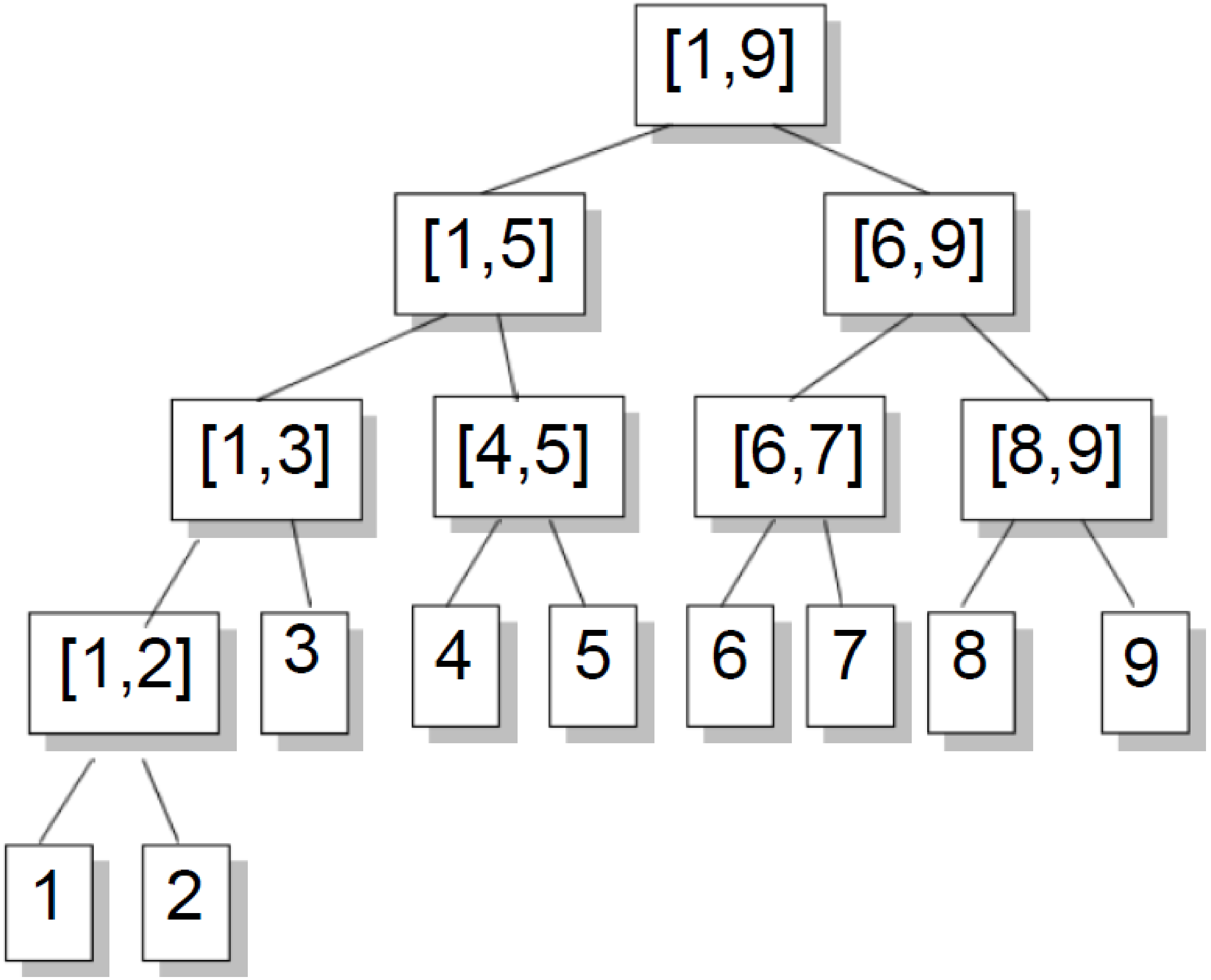
区间[1, 10]对应的线段树



- 同一层结点所代表的区间，相互不会重叠。
- 除最下一层外，同一层结点所代表的区间加起来是连续的区间。
- 除了最后一层，其他层构成一棵满二叉树。

线段树（Segment Tree）

区间[1, 9]对应的线段树



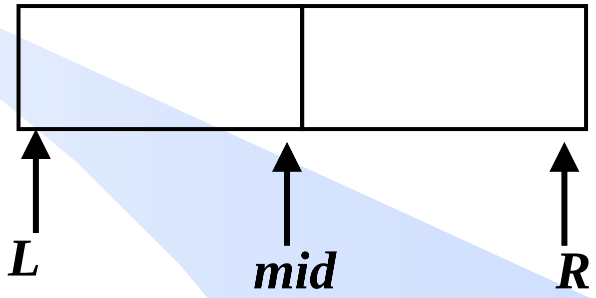
- 若根结点对应的区间是[1,*n*]，则树高O(log*n*)
- 叶结点的数目和根结点表示的区间长度相同。结点度0或2，叶结点*n*个，总结点2*n*-1个。
- 在结点内，可以维护关于区间的信息，如和、最值等。
- 可用顺序存储，数组开4*n*大小。

```
struct Node{
    int L,R;
    int sum;
};
Node tree[N*4];
```

（1）建树

```
struct Node{
    int L,R;
    int sum;
};
Node tree[N*4];
```

```
void build(int root, int L, int R){
   //针对区间[L, R]构建一棵以root为根的线段树
   tree[root].L = L; tree[root].R = R;
   if (L == R) {   //叶结点，区间长度为1
      tree[root].sum = a[L];  return;
   }
   int mid = (L + R)/2;
   build(2*root, L, mid); //针对区间[L, mid]构建root的左子树
   build(2*root+1, mid+1, R); //针对区间[mid+1, R]构建root的右子树
   tree[root].sum=tree[2*root].sum+tree[2*root+1].sum;
}
```

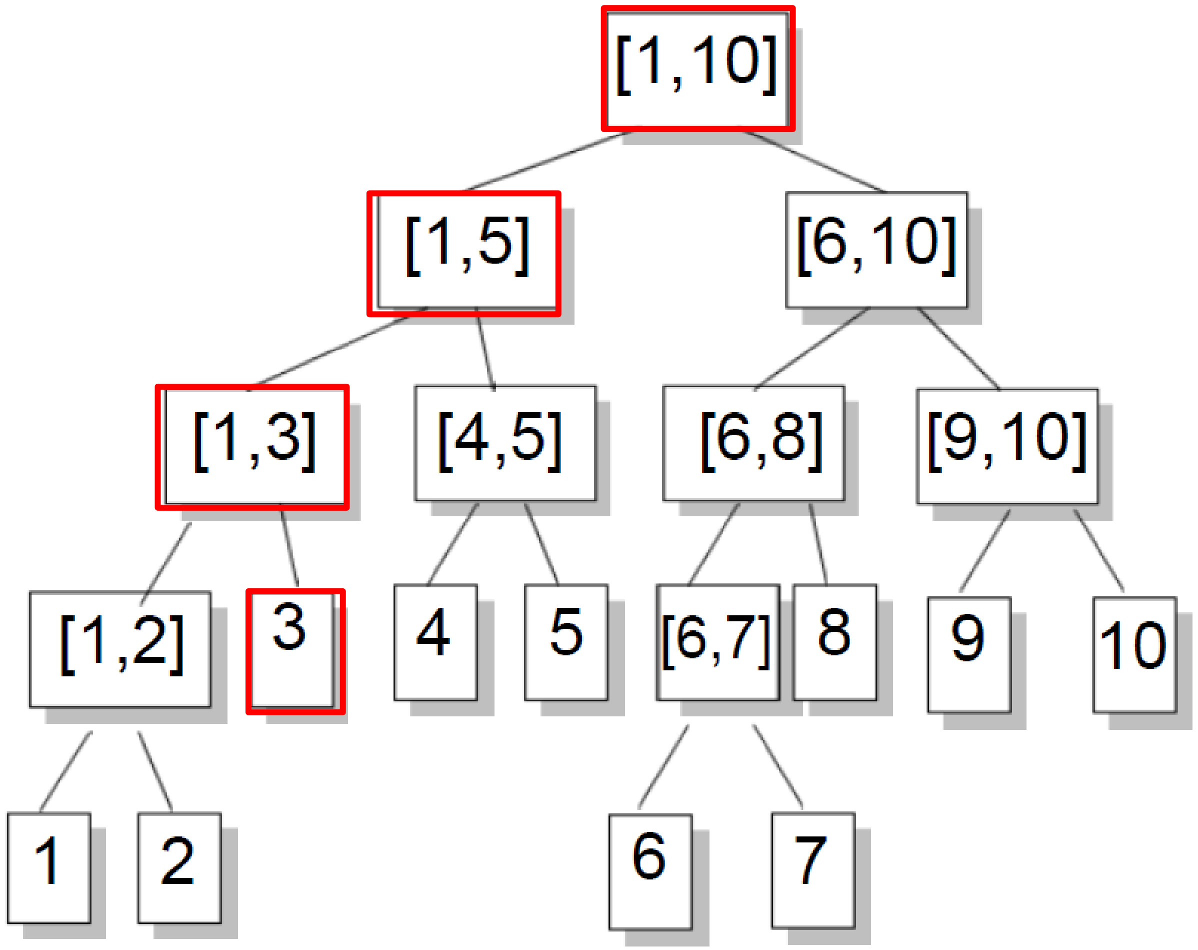


针对区间[1,*n*]建一棵线段树，初始调用build(1, 1, n)

时间O(*n*)

（2）单点更新

操作：a[3]+=x



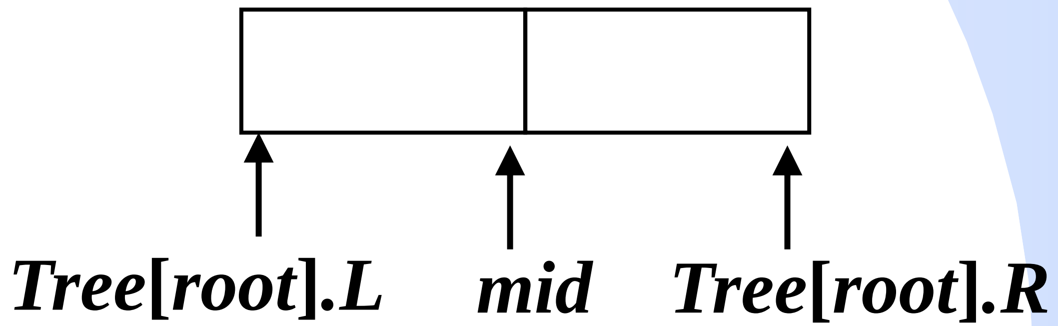
从根结点向下搜索，查找包含3的区间对应的结点，其sum域均应+=x

```
struct Node{
    int L,R;
    int sum;
};
```

（2）单点更新

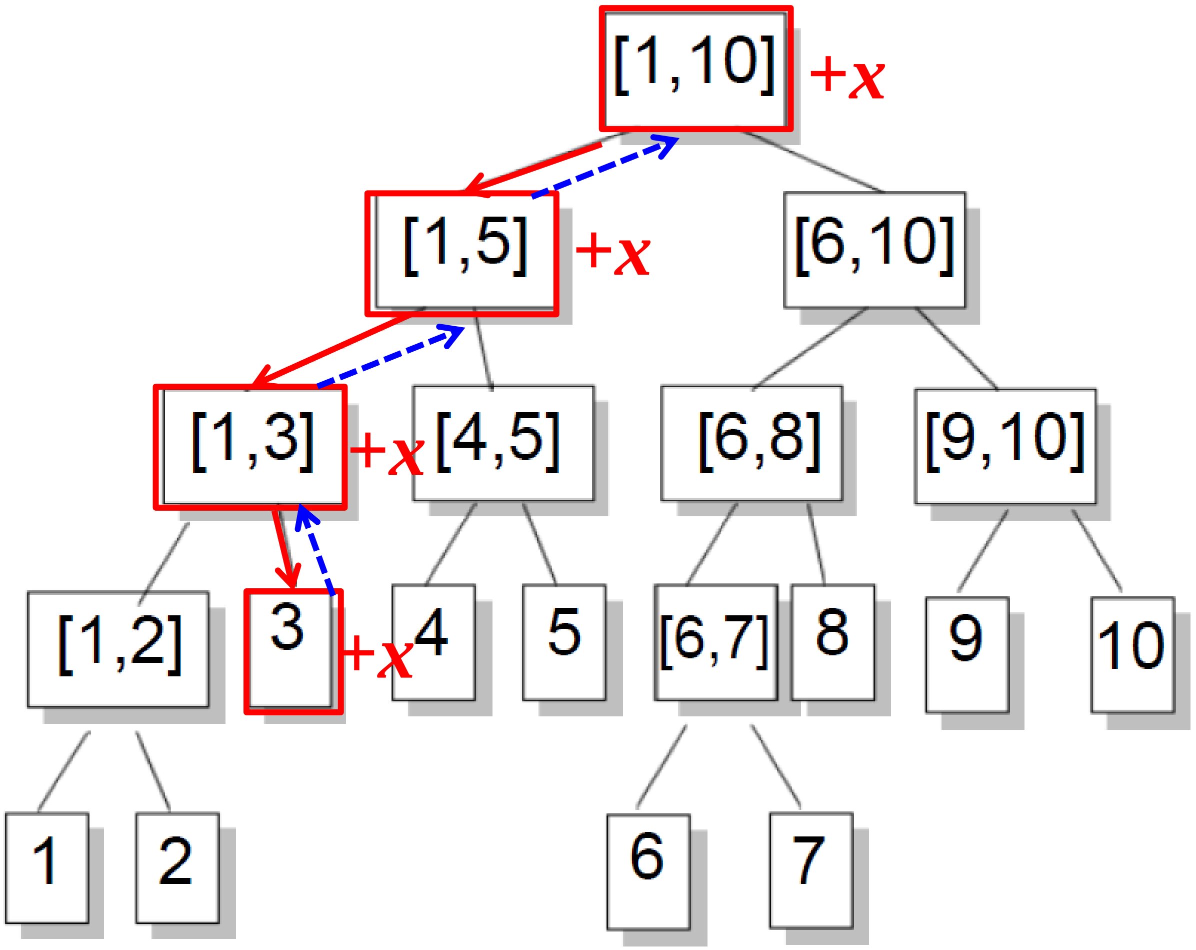
```
void update(int root, int i, int x){ /*把a[i]加上x, 即在以root
   为根的线段树中搜索包含i的区间对应的结点, 将其sum域都加上x */
   if (tree[root].L==tree[root].R){//搜索到叶结点, 其对应区间为[i]
     tree[root].sum += x; return;  //该区间只包含一个元素a[i]
   }
   int mid = (tree[root].L + tree[root].R)/2;
   if(i <= mid) update(2*root, i, x);  //在root的左子树中搜索
   else update(2*root+1, i, x);        //在root的右子树中搜索
   tree[root].sum = tree[2*root].sum+tree[2*root+1].sum;
}
```

时间复杂度
O(logn)



（2）单点更新

操作：a[3]+=x



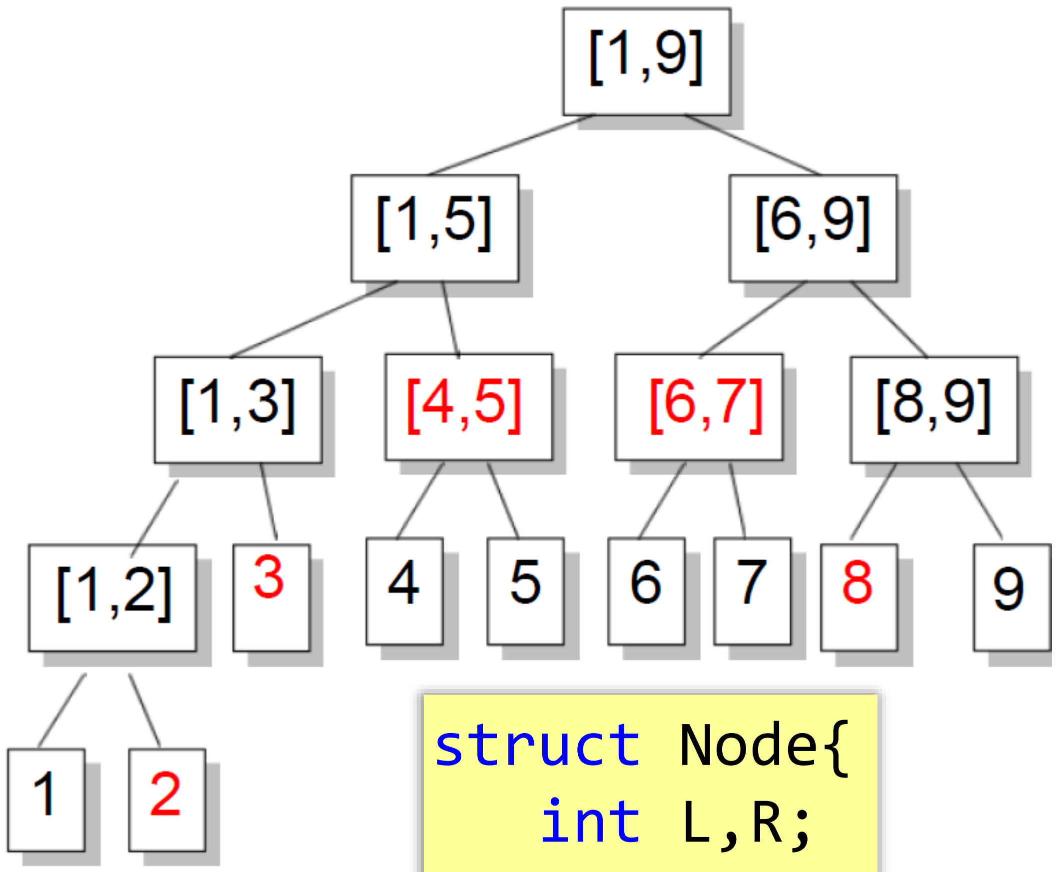
从根结点搜索包含3的区间对应的结点，其sum域均应+=x

时间复杂度

O(log*n*)

（3）区间查询

查询：区间[2,8]的和



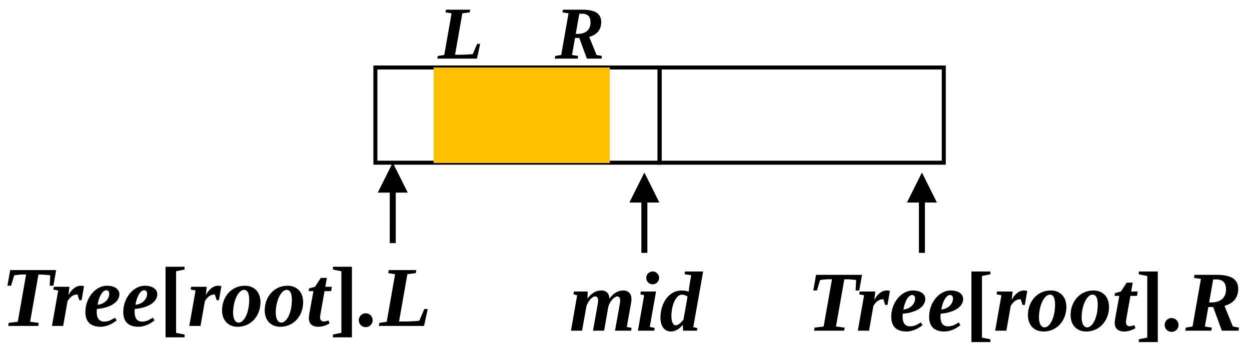

```
struct Node{
    int L,R;
    int sum;
};
```

在树中找出一些结点，这些结点对应的区间相互不重叠，且加起来正好是[2,8]，这些结点称为终止结点。

从根结点开始递归查询。走到结点[L,R]时， 如果查询的区间就是[L,R]，则找到一个终止结点；如果不是，则看要查询的区间与左子树/右子树对应的区间哪个有交集，就进入个区间进一步查询。

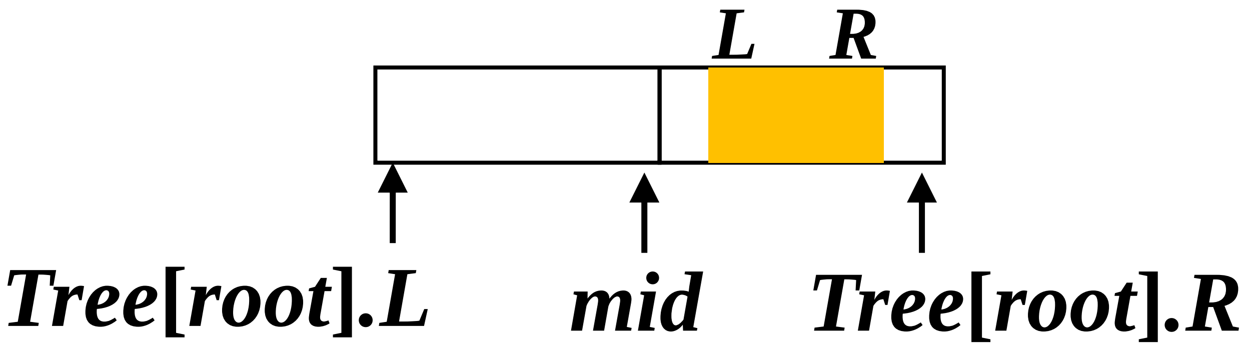
(3) 区间查询

```
int query(int root, int L, int R){
  //在以root根的线段树中查询区间[L, R]的和
  if (L==tree[root].L && R==tree[root].R)
      return tree[root].sum;  //若根结点对应的区间即为[L, R]
  int mid=(tree[root].L+tree[root].R)/2;
  if (R<=mid) return query(2*root, L, R);
  //若区间[L, R]在当前结点左部，则在左子树中查询区间[L, R]的和
```



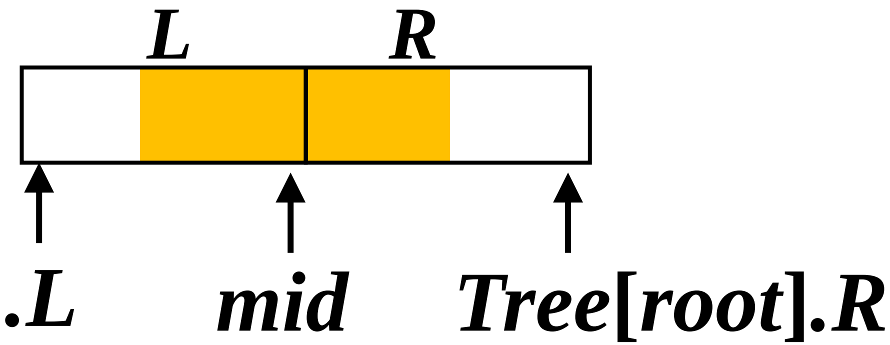
（3）区间查询

```
int query(int root, int L, int R){
  //在以root根的线段树中查询区间[L, R]的和
   if (L==tree[root].L && R==tree[root].R)
       return tree[root].sum;  //若根结点对应的区间即为[L, R]
   int mid=(tree[root].L+tree[root].R)/2;
   if (R<=mid) return query(2*root, L, R);
   else if (L>mid) return query(2*root+1, L, R);
   //若区间[L, R]在当前结点右部，则在右子树中查询区间[L, R]的和
```

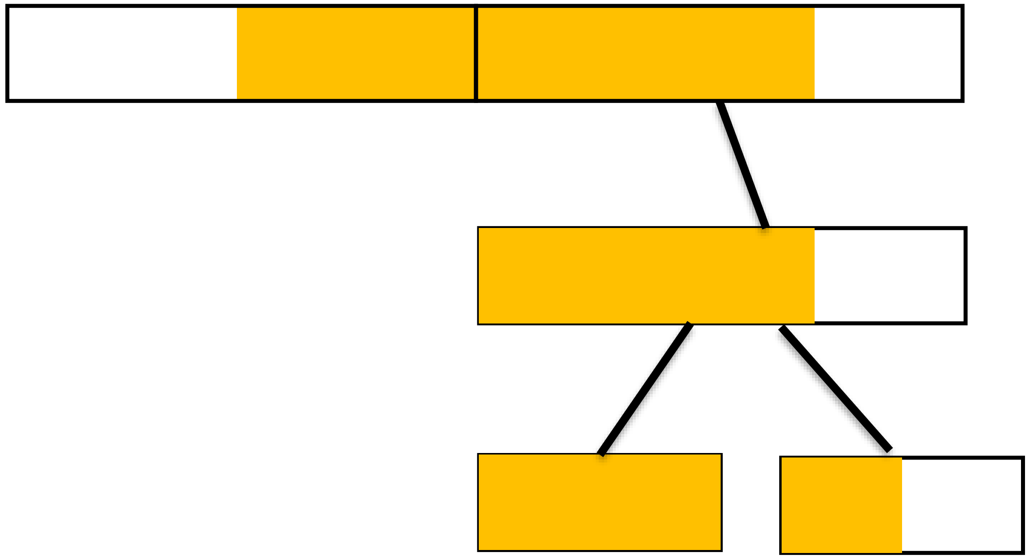


(3) 区间查询

```
int query(int root, int L, int R){
  //在以root根的线段树中查询区间[L, R]的和
   if (L==tree[root].L && R==tree[root].R)
       return tree[root].sum;  //若根结点对应的区间即为[L, R]
   int mid=(tree[root].L+tree[root].R)/2;
   if (R<=mid) return query(2*root, L, R);
   else if (L>mid) return query(2*root+1, L, R);
   else return query(2*root,L,mid)+query(2*root+1,mid+1,R);
}
```



时间复杂度
O(logn)



区间问题处理技巧

- 前缀和
- 差分数组
- ST表
- 尺取法
- 线段树入门
- **树状数组入门**

算法之道 结构之美 数据之法

zhuyungang@jlu.edu.cn

位运算

- ✓ 5&3=1
- ✓ 6&(-6) = 0110&1010 = 0010
- ✓ *x* & (-*x*) = *x*的二进制表示形式
  留下最右边的1, 其他位都变成0
- ✓ *lowbit*(*x*) = *x* & (-*x*)

```
int lowbit(int x){
    return x & -x;
}
```

```
      x  = +++++100
     -x  = -----011 +1
         = -----100
x&(-x)   = 00000100
```

lowbit(1) = *lowbit*(0001) = 1
lowbit(2) = *lowbit*(0010) = 2
lowbit(3) = *lowbit*(0011) = 1
lowbit(4) = *lowbit*(0100) = 4
lowbit(5) = *lowbit*(0101) = 1
lowbit(6) = *lowbit*(0110) = 2
lowbit(7) = *lowbit*(0111) = 1
lowbit(8) = *lowbit*(1000) = 8

树状数组 (Binary Indexed Tree)

- ✓ 对于数组a，我们设一个数组c
- ✓ c[i]是a的连续若干个元素之和，其中最后一个元素是a[i]。
- ✓ $c[i] = \underbrace{a[i-\mathrm{lowbit}(i)+1] + a[i-\mathrm{lowbit}(i)+2] + \ldots + a[i]}_{\mathrm{lowbit}(i)\text{个}}$
- ✓ i从1开始算，c[0]和a[0]没用
- ✓ c即为a的树状数组

$$c[i]=a[i-\mathrm{lowbit}(i)+1] + a[i-\mathrm{lowbit}(i)+2] + \ldots + a[i]$$

$c1 = a1$

$c2 = a1+a2$

$c3 = a3$

$c4 = a1+a2+a3+a4$

$c5 = a5$

$c6 = a5+a6$

$c7 = a7$

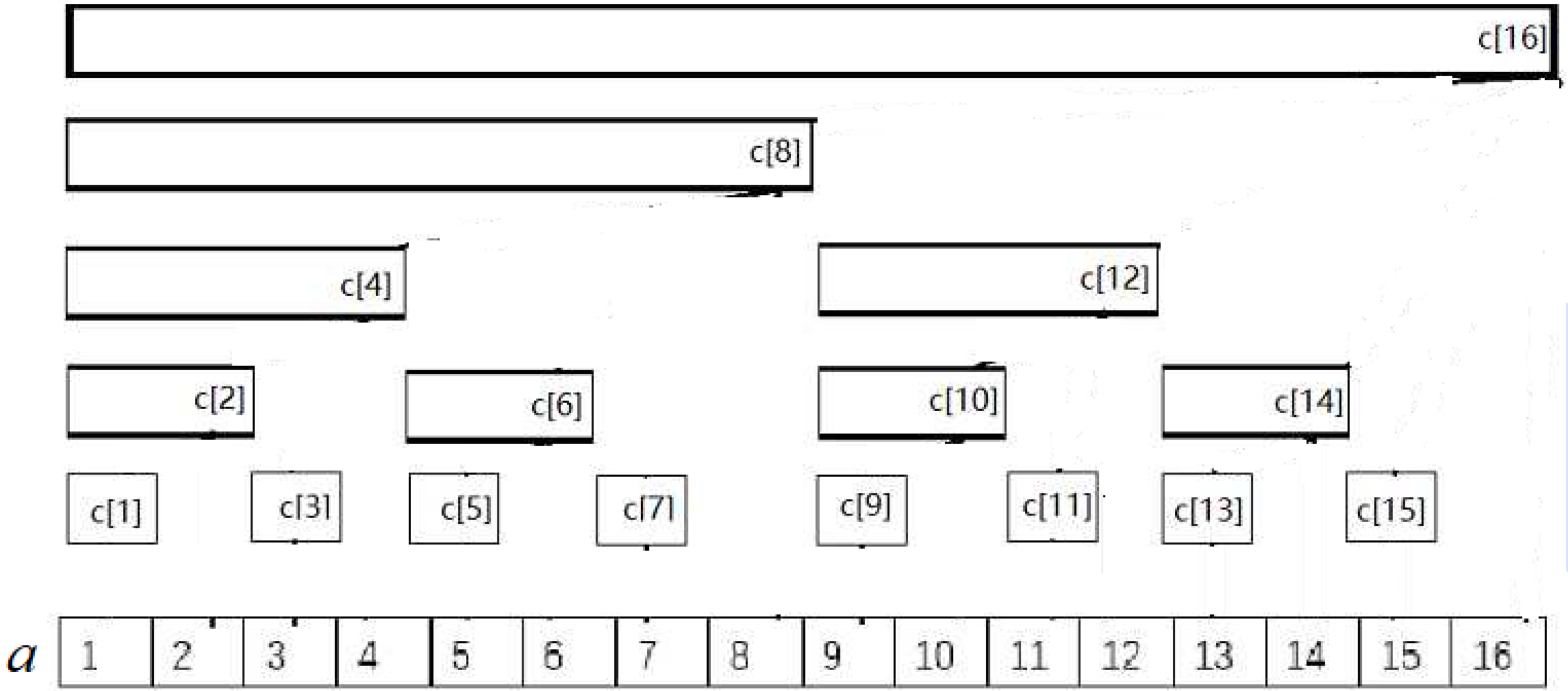
$c8 = a1+a2+a3+a4+a5+a6+a7+a8$

$\ldots\ldots\ldots\ldots$

$c16 = a1+a2+a3+a4+a5+a6+a7+a8+a9+a10+a11+$
$a12+a13+a14+a15+a16$

$lowbit(1)=lowbit(0001)=1$
$lowbit(2)=lowbit(0010)=2$
$lowbit(3)=lowbit(0011)=1$
$lowbit(4)=lowbit(0100)=4$
$lowbit(5)=lowbit(0101)=1$
$lowbit(6)=lowbit(0110)=2$
$lowbit(7)=lowbit(0111)=1$
$lowbit(8)=lowbit(1000)=8$

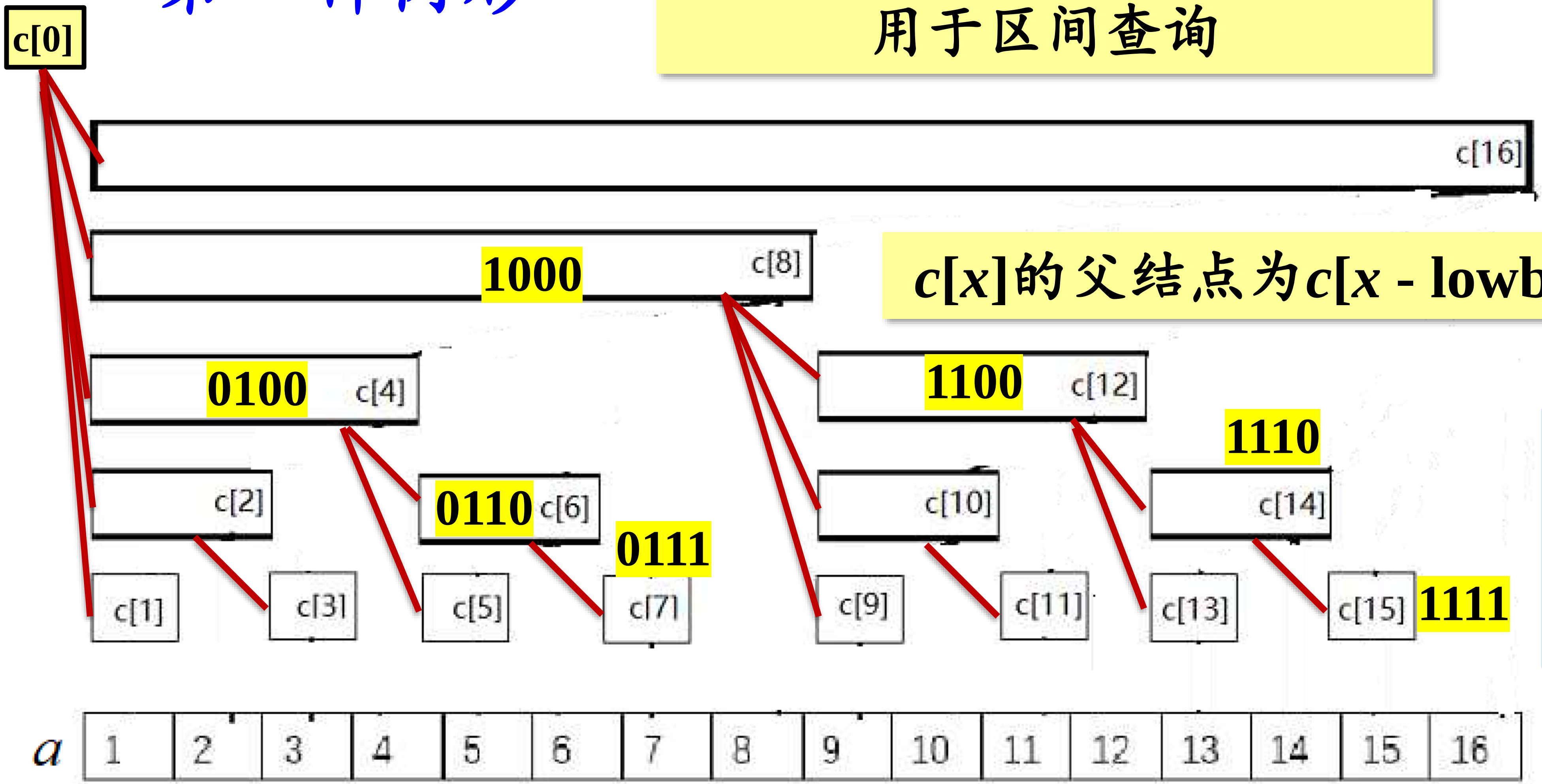
树状数组示例



c[16]
c[8]
c[4]
c[12]
c[2]
c[6]
c[10]
c[14]
c[1]
c[3]
c[5]
c[7]
c[9]
c[11]
c[13]
c[15]
a
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

第一种树形

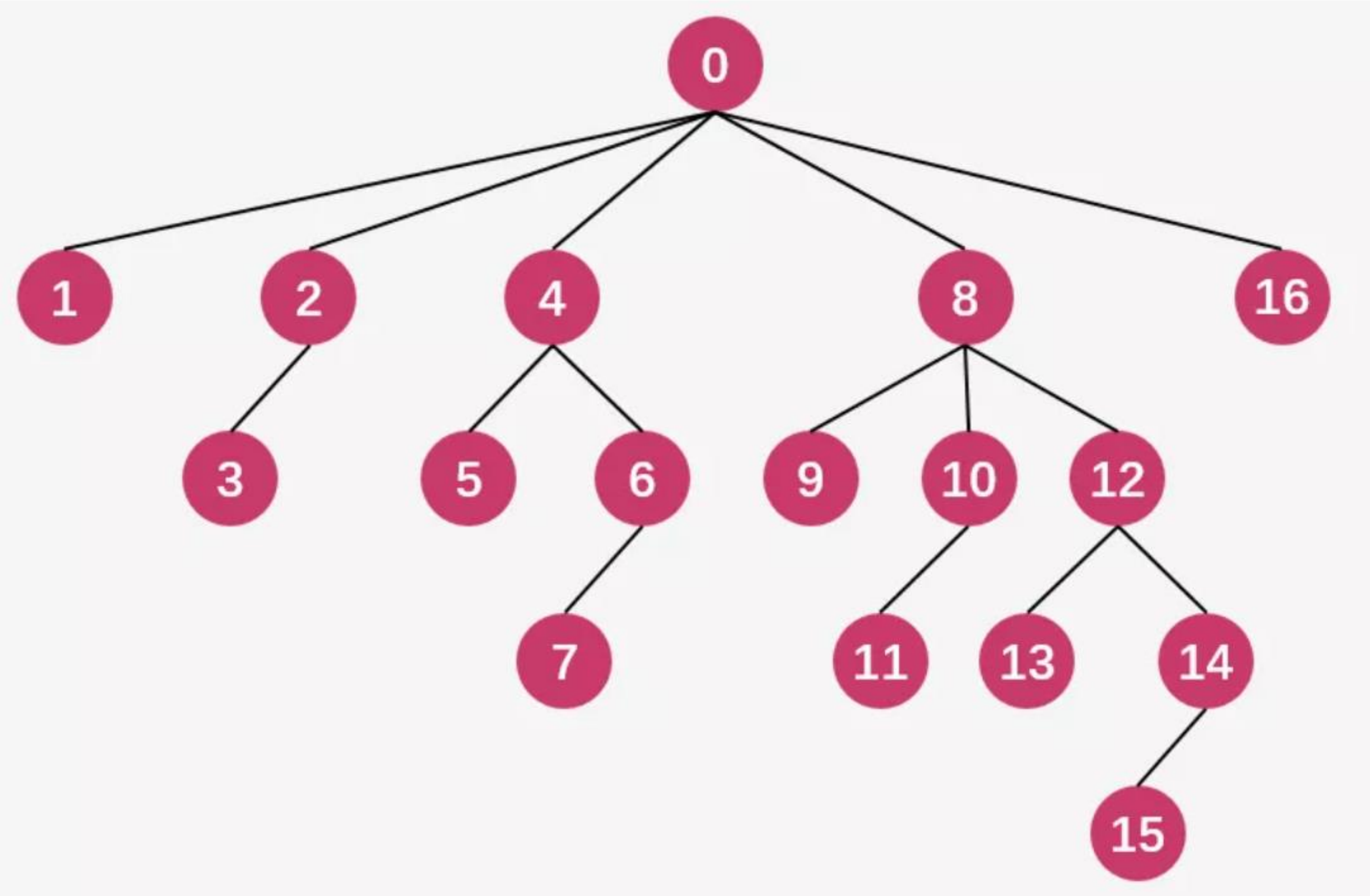
父子关系：区间的相邻关系
用于区间查询



*x*的父结点：*x*对应的二进制数减去最右边的1，即*x* - lowbit(*x*)

查询*a*[1]+...+*a*[*i*]：从c[*i*]开始沿父结点往上走，将沿途结点累加，即将c[*i*]及其祖先结点相加。

第一种树形



（1）区间查询

```
int query(int c[],int i) { //查询a[1]+…+a[i]
    for(int sum=0; i>0; i-=lowbit(i))
        sum += c[i];
    return sum;
}
```

c[*i*]的父结点为*c*[*i*-lowbit(*i*)]

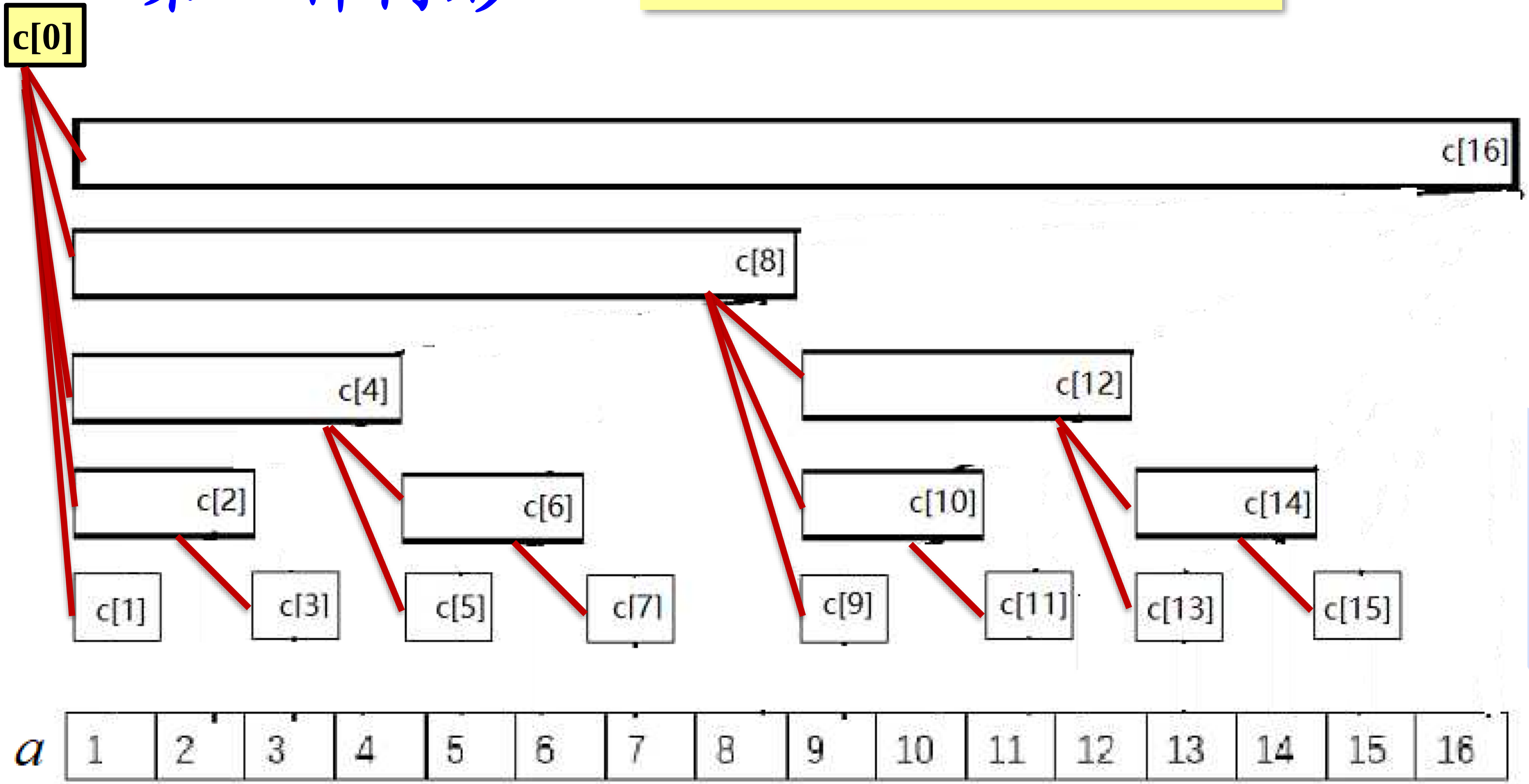
查询*a*[1]+…+*a*[*i*]：从*c*[*i*]开始沿父结点往上走，将沿途结点累加，即将*c*[*i*]及其祖先结点相加。

查询区间[*L*,*R*]的和*a*[*L*]+…+*a*[*R*]
query(c,R)-query(c,L-1)

时间O(log*n*)

第一种树形

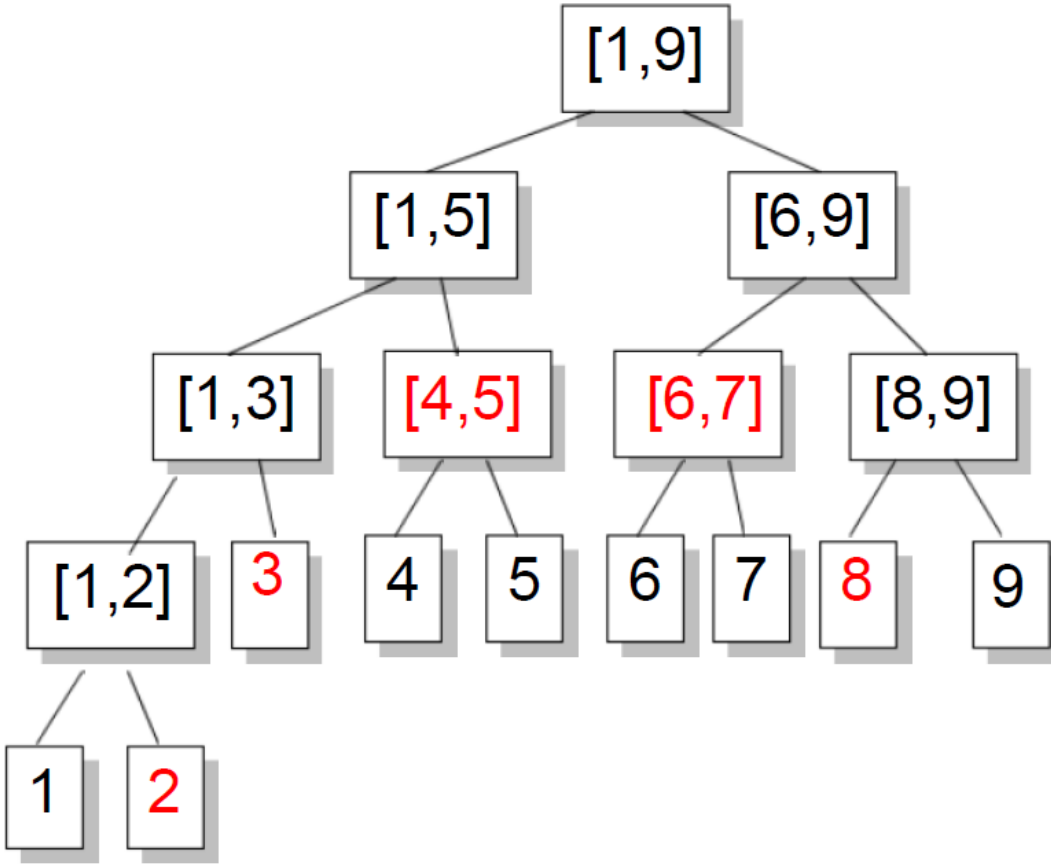
本质：分块思想



查询a[1]+a[2]+a[3]+a[4]+a[5]+a[6]+a[7]

回顾：线段树

查询：区间[2,8]的和



将查询区间分解成若干小区间，小区间个数O(log*n*)

（2）单点更新

$c1 = a1$

$c2 = a1+a2$

$c3 = a3$

$c4 = a1+a2+a3+a4$

$c5 = a5$

$c6 = a5+a6$

$c7 = a7$

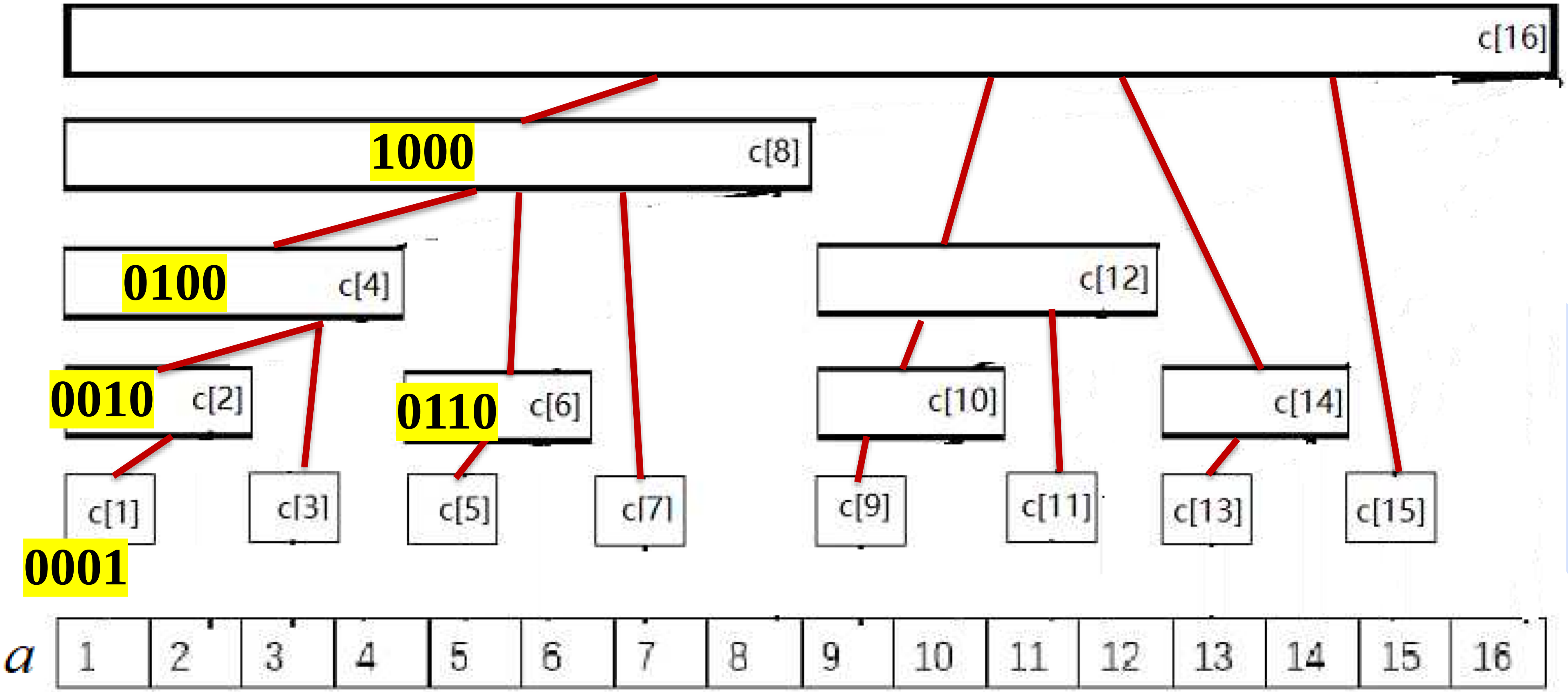
$c8 = a1+a2+a3+a4+a5+a6+a7+a8$

……………

$c16 = a1+a2+a3+a4+a5+a6+a7+a8+a9+a10+a11+$
$a12+a13+a14+a15+a16$

第二种树形

父子关系：区间的包含关系
用于单点更新



c[x]的父结点为
c[x+lowbit(x)]

更新a[i]：从c[i]开始沿父结点往上走，将沿途结点更新。

(2) 单点更新

```
void update(int c[],int i, int x) {  //a[i]+=x
    for(; i<=n; i+=lowbit(i))
        c[i]+=x;
}
```

c[*i*]的父结点为c[*i*+lowbit(*i*)]

时间O(log*n*)

更新*a*[*i*]：从*c*[*i*]开始沿父结点往上走，将沿途结点更新，即将*c*[*i*]及其祖先结点更新。

（3）构建树状数组

$c[i]=a[i-\mathrm{lowbit}(i)+1]+a[i-\mathrm{lowbit}(i)+2]+\ldots+a[i]$

```
const int maxn=1e5+10;
void build(int a[], int c[], int n){
    int sum[maxn];
    sum[0]=0;
    for(int i=1; i<=n; i++){
        sum[i]=sum[i-1]+a[i];
        c[i]=sum[i]-sum[i-lowbit(i)];
    }
}
```

时间复杂度
$O(n)$

应用：海量用户的高性能、低延迟实时排行榜

- 千万级用户，用户积分经常变化，用户可查询自己的实时排名。常规做法：O(n)及以上。
- 数组a的下标表示积分，$a[i]$表示拥有积分i的用户个数，例如$a[100]=9$，表示积分为100的用户有9个。初始时基于a构建树状数组c。
- 当更新某用户的积分时，假设将积分x更新为积分y，可将$a[x]$-=1，$a[y]$+=1（单点更新）。
- 查询某用户排名时，假定该用户积分为x，则a的前缀和$sum[x]$（区间求和）表示积分小于等于x的用户个数，假定总用户为n，则该用户排名为$n-sum[x]+1$。

树状数组 vs 线段树

- ✓ 树状数组能解决的问题线段树都能解决，线段树能解决的问题树状数组未必能解决。
- ✓ 线段树和树状数组时间复杂度相同，但树状数组的常数更低些，且空间消耗更少，代码简单。
- ✓ 如果一个问题既能用树状数组也能用线段树解决，首选树状数组。
- ✓ 单点更新区间求和，树状数组更快。

单点更新、区间求和问题

	暴力方法	前缀和	线段树 树状数组
单点更新	O(1)	O(*n*)	O(log*n*)
区间求和	O(*n*)	O(1)	O(log*n*)
*n*次更新 *n*次查询总时间	O(n^2)	O(n^2)	O(*n*log*n*)
单次更新/查询 均摊时间	O(*n*)	O(*n*)	O(log*n*)