中国计算机学会
China Computer Federation

分治算法及其应用

安徽师大附中　　叶国平

QQ:17412182

讲课的主要内容

- 1. 分治思想
- 2. 经典问题
- 3. CDQ分治
- 4. 整体二分
- 5. 点分治

分治思想

- 分治(Divide-And-Conquer)就是“分而治之”的意思，其实质就是将原问题分成n个规模较小而结构与原问题相似的子问题；然后递归地解这些子问题，最后合并其结果就得到原问题的解。
- 其三个步骤如下：

①划分问题(Divide)：将原问题分成一系列子问题。

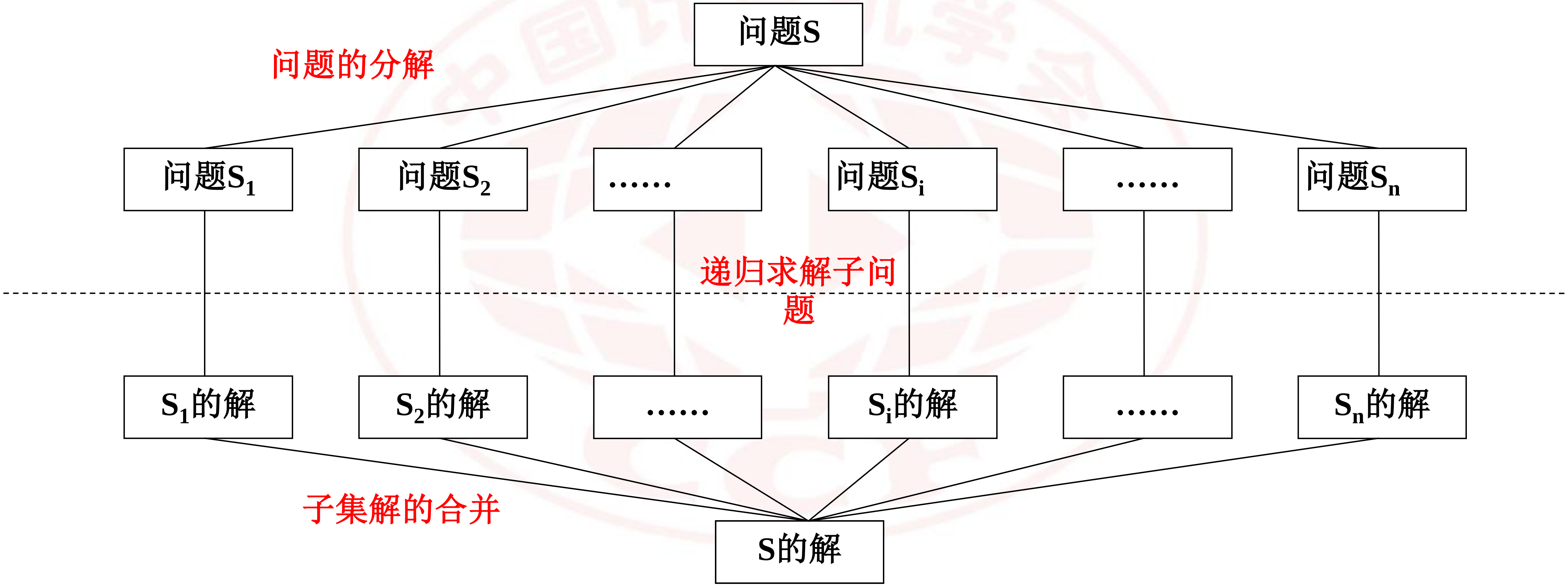
②递归解决(Conquer)：递归地解各子问题。若子问题足够小，则可直接求解。

③合并问题(Combine)：将子问题的结果合并成原问题的解。

适用条件

- 采用分治法解决的问题一般具有的特征如下:
  - 1. 问题的规模缩小到一定的规模就可以较容易地解决。
  - 2. 问题可以分解为若干个规模较小的模式相同的子问题，即该问题具有子结构性质。
  - 3. 合并问题分解出的子问题的解可以得到问题的解。
  - 4. 问题所分解出的各个子问题之间是独立的，即子问题之间不存在公共的子问题。

分治算法设计过程图



问题S
问题的分解
问题S_1
问题S_2
……
问题S_i
……
问题S_n
递归求解子问题
S_1的解
S_2的解
……
S_i的解
……
S_n的解
子集解的合并
S的解

主定理

- 递归算法在分析复杂度时，有专门的结论可以使用，而这个结论称之为主定理：规模为n的问题通过分治，得到a个规模为 $\frac{n}{b}$ 的子问题，每次递归带来的额外计算(即除子问题之外的计算工作)量为cn^d,则处理规模为n的问题的时间为T(n)有：$T(n) = aT(\frac{n}{b}) + cn^d$
  - 若$\boldsymbol{a = b^d}$,则$T(n) = \boldsymbol{O}(cn^d logn)$
  - 若$\boldsymbol{a < b^d}$,则$T(n) = \boldsymbol{O}(cn^d)$
  - 若$\boldsymbol{a > b^d}$,则$T(n) = \boldsymbol{O}(cn^{logb^a})$

中国计算机学会 China Computer Federation

经典应用

- 二分查找
- 快速排序
- 归并排序
- 大整数乘法
- 棋盘覆盖
- 最近点对问题
- 矩阵乘法
- ……

中国计算机学会 China Computer Federation

快速排序

- 基本策略
  - 把要排序的数据数组分成两个子数组：
  - 在第一个子数组中的数据比一个已知值要小
  - 在第二个数组中的数据比那个值要大
  - 技术上来说称为“分块”
- 已知值称为‘基准元素’或称‘枢元素’
  - 一旦我们已经分好块，基准元素将处于它最终的位置
  - 然后我们继续把子数组分为更小的数组，直到剩余部分只有一个元素
  - 可以递归实现

中国计算机学会 China Computer Federation

快速排序

- 1. 选择一个元素作为枢元素（随机选取）。
- 2. 在左边和右边元素开始索引
- 3. 移动左边的索引直到我们得到一个元素>枢元素
- 4. 移动右边的索引直到我们得到一个元素<枢元素
- 5. 若索引不相交，则交换值并重复步骤3和4
- 6. 若索引相交，则交换枢元素值和左边索引值
- 7. 在子数组调用快速排序得到枢元素左右的值

实例

Quicksort(a,0,6)

原始值	45	80	55	85	50	70	65
	↑ i						↑ j
第一次交换后	45	50	55	85	80	70	65
		↑ i			↑ j		
第二次交换后	45	50	55	65	80	70	85

Quicksort(a,0,2) Quicksort(a,4,6)

中国计算机学会 China Computer Federation

程序

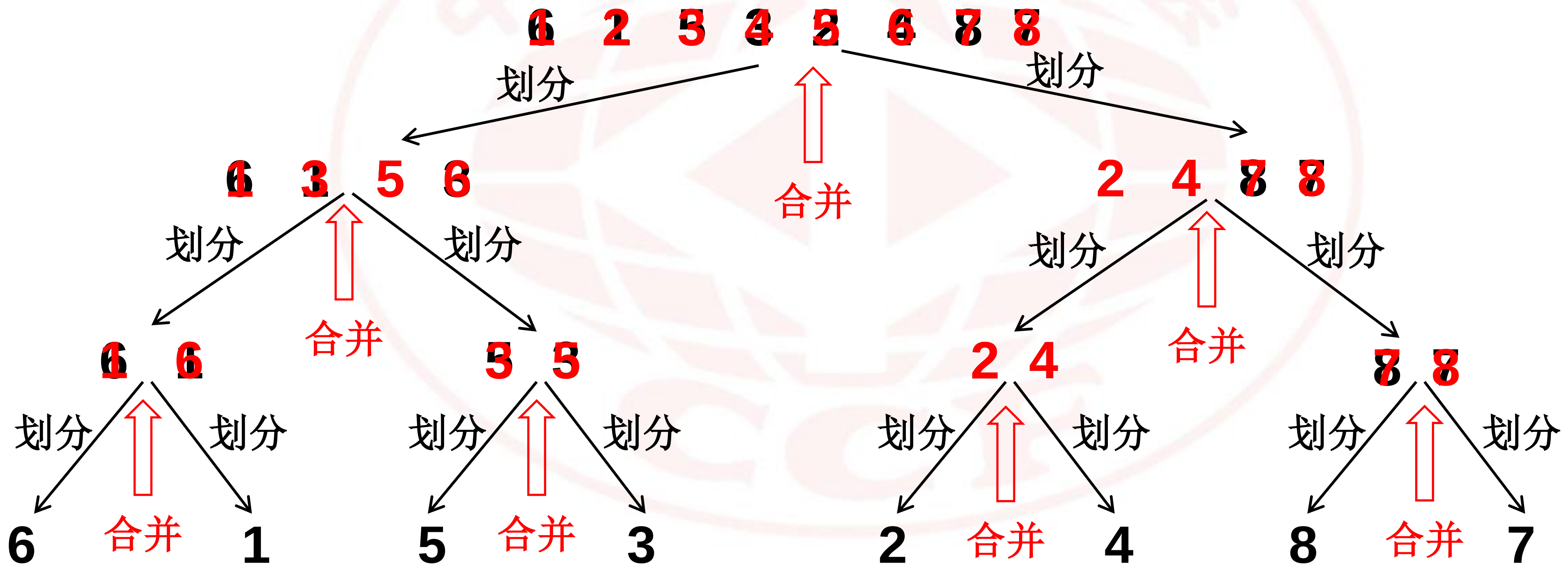
```
void quick_sort(int a[ ], int l, int r)
{if (l < r)
   {int i = l, j = r, x = a[l];  //选择第一个数作为基准元素
    while (i < j)
      {while(i < j && a[j] >= x) j--; //从右向左找第一个<x的数
       if (i < j) a[i++] = a[j];
       while(i < j && a[i] < x) i++;  //从左向右找第一个>=x的数
       if (i < j) a[j--] = a[i];
      }
    a[i] = x;
    quick_sort(a, l, i - 1);  // 递归调用
    quick_sort(a, i + 1, r);
   }
}
```

归并排序

- 给定一个长度为N的序列，对其进行排序。
- 归并排序思想：
  - 1. 划分问题：把长度为N的序列尽可能均分成左右两部分；
  - 2. 递归求解：对左右两部分分别进行归并排序；
  - 3. 合并问题：把左右两段排好序的序列合并出最终的有序序列。

归并排序

➢ 举例：N=8, 8个数分别为6, 1, 5, 3, 2, 4, 8, 7。过程如下：



归并排序

- 第3步如何把两段排好序的序列合并成一个有序序列?
- 考虑第一段中两个数A[i],A[j]，如果i<j，则A[i] 肯定比A[j] 要早出现
- 策略：比较两个序列最小的数, 谁小谁在前; 重复比较直到没有数为止。时间复杂度为O(n)。

归并排序

```
void merge(l,m,r)  // 合并操作
  {// i 代表左半序列目前最小的位置
   // j 代表右半序列目前最小的位置
   // a：原数组b 数组：暂存数组（合并过程不能覆盖原数组）
   int i = l, j = m+1, k = l;
   for(; i<=m && j<=r; )
     if (a[i] <= a[j]) b[k++] = a[i++];  // 加入左边最小的
     else b[k++] = a[j++]; // 加入右边最小的
   // 加入剩余的数
   for(; i<=m; b[k++] = a[i++]);  // 加入剩余左半的数
   for(; j<=r; b[k++] = a[j++]);  // 加入剩余右半的数
   for(i=l; i<=r; i++)  a[i] = b[i];  // 从暂存数组中赋值
  }
```

归并排序

```
void merge_sort(int l, int r) // 排序位置从l 到r 的序列
    {
        if (l==r) return; //长度为1 则不需要排序
        int m = (l+r) / 2;  // 分成两段
        merge_sort(l, m); // 排序左半段
        merge_sort(m+1, r); // 排序右半段
        merge(l,m,r); //把两段排好序的序列合并成一个有序序列
    }
```

归并排序

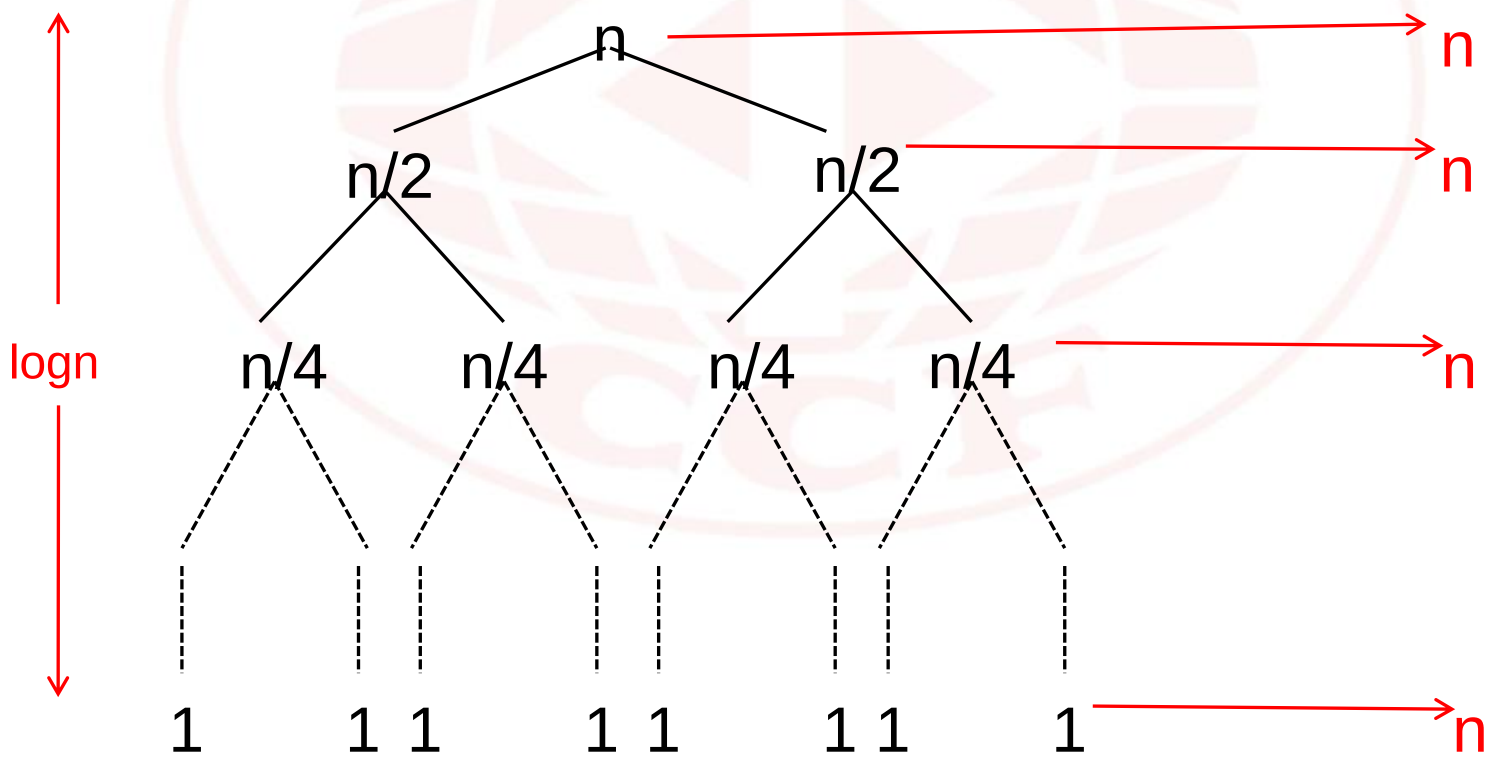
- T(n)表示对n个数进行归并排序的时间复杂度
- 根据归并排序思想有：T(n)=2*T(n/2)+n
- 设m=2^k, k=logm,则有：
- T(m)=2*T(m/2)+m=2*(2*T(m/4)+m/2)+m=4*T(m/4)+2*m

  =4*(2*T(m/8)+m/4)+2*m=8*T(m/8)+3*m

  =……=2^k*T(1)+k*m=2^k+k*2^k=m+m*logm=mlogm
- 设m<=n<2*m，则有T(m)<=T(n)<T(2*m)
- m+mlogm<=T(n)<2*m+2*m*log(2*m)=2*m+2*m+2*m*logm
- O(mlogm)<=T(n)<=O(2*m*logm)
- 所以T(n)=O(nlogn)。

归并排序

- 画出归并排序的递归树。
- 时间主要花费在合并上，每一层合并代价综合为n,树的深度为logn，所以时间复杂度为O(nlogn)。



棋盘覆盖

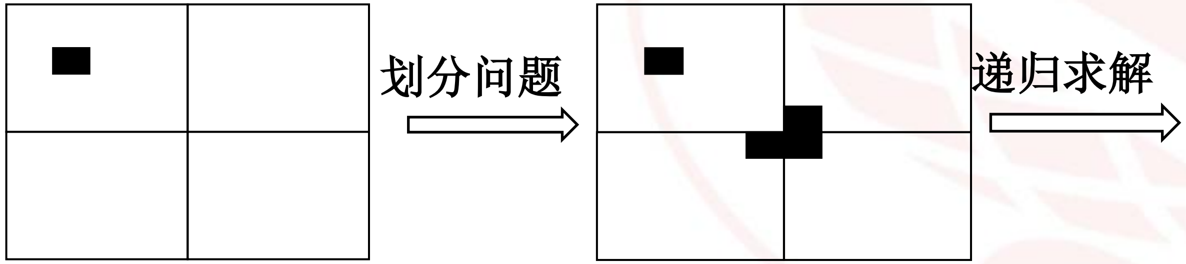
- 有一个2^k*2^k的方格棋盘，恰有一个方格是黑色的，其他为白色。你的任务是用包含3个方格的L型牌覆盖所有白色方格。黑色方格不能被覆盖，且任意一个白色方格不能同时被两个或更多牌覆盖。如下图所示为L型牌的4种旋转方式。

棋盘覆盖

- 题目是要求用L型牌覆盖一个2^k*2^k的方格棋盘，棋盘有且只有一个黑色方格，且此方格不能被覆盖。
- 用分治来解决，把原棋盘平均分为4块，通过用1个L型牌覆盖棋盘正中间2*2的区域中的三个格子，把原棋盘分成4块$2^{k-1}*2^{k-1}$的方格棋盘,且每块中只含有一个黑色方格(不能覆盖的格子)。
- 此时把原问题分解成4个结构与原问题一样的子问题，棋盘的大小(边长)每次减半，最终会出现边长为1的情况，直接求解即可。

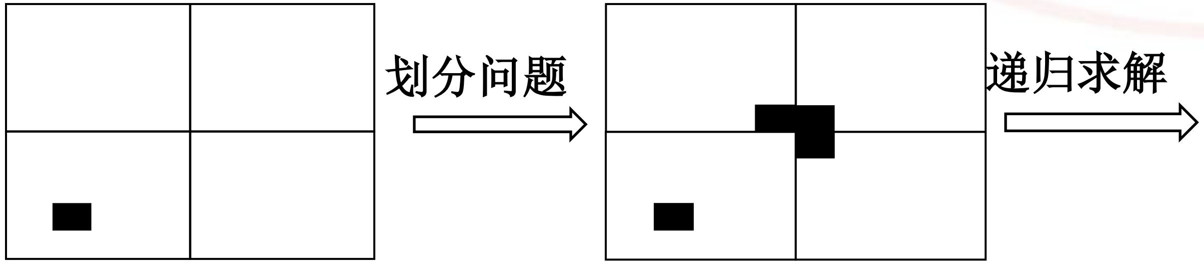
棋盘覆盖

- 定义cover(x,y,len,x0,y0)表示求解左上角格子在(x,y)边长为len,其中(x0,y0)不能覆盖的棋盘覆盖方案。
- 根据黑色格子的位置分为以下四种情况：
- 情况1：黑点位于左上部分



左上:cover(x,y,len/2,x0,y0)
左下:cover(x+len/2,y,len/2,x+len/2,y+len/2-1)
右上:cover(x,y+len/2,len/2,x+len/2-1,y+len/2)
右下:cover(x+len/2,y+len/2,len/2,x+len/2,y+len/2)

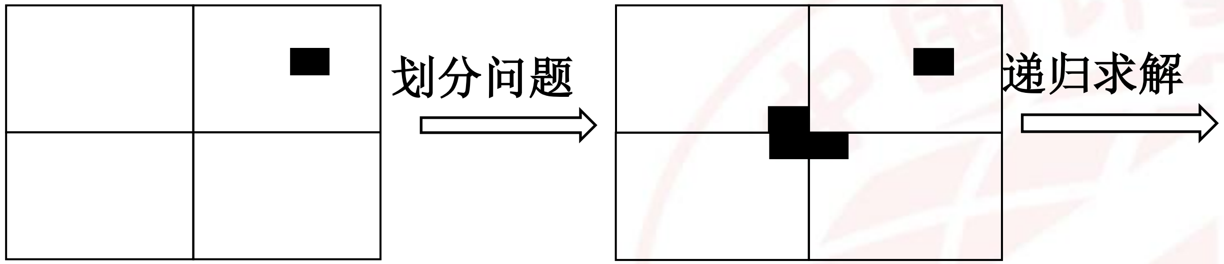
- 情况2：黑点位于左下部分



左上:cover(x,y,len/2,x+len/2-1,y+len/2-1)
左下:cover(x+len/2,y,len/2,x0,y0)
右上:cover(x,y+len/2,len/2,x+len/2-1,y+len/2)
右下:cover(x+len/2,y+len/2,len/2,x+len/2,y+len/2)

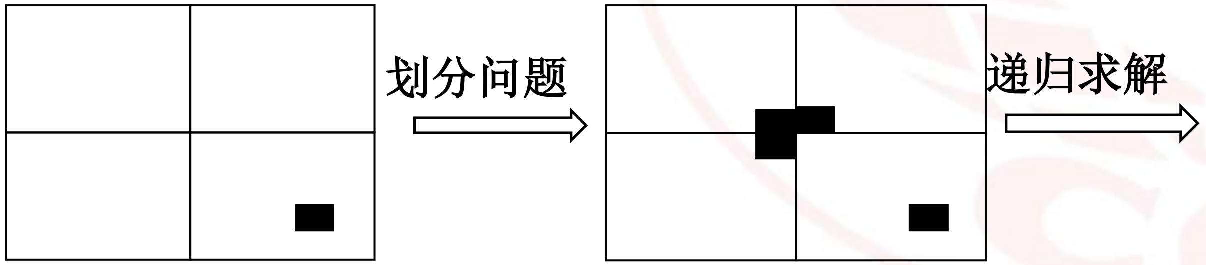
棋盘覆盖

- 情况3：黑点位于右上部分



左上:cover(x,y,len/2, x+len/2-1,y+len/2-1)
左下:cover(x+len/2,y,len/2,x+len/2,y+len/2-1)
右上:cover(x,y+len/2,len/2,x0,y0)
右下:cover(x+len/2,y+len/2,len/2,x+len/2,y+len/2)

- 情况4：黑点位于右下部分



左上:cover(x,y,len/2,x+len/2-1,y+len/2-1)
左下:cover(x+len/2,y,len/2,x+len/2,y+len/2-1)
右上:cover(x,y+len/2,len/2,x+len/2-1,y+len/2)
右下:cover(x+len/2,y+len/2,len/2,x0,y0)

- 每个格子只会被处理一次，时间复杂度为O(n^2)，n为棋盘边长。

棋盘覆盖

```
void cover(int x, int y, int len, int x0, int y0)
{   int t,len1;
    if (len==1)return;
    t=++tot;
    len1=len/2;
    if (x0<x+len1&&y0<y+len1)  cover(x,y,len1,x0,y0);
   else{board[x+len1-1][y+len1-1]=t; cover(x,y,len1,x0+len1-1,y0+len1-1);}
    if (x0>=x+len1&&y0<y+len1)  cover(x+len1,y,len1,x0,y0);
    else{board[x+len1][y+len1-1]=t; cover(x+len1,y,len1,x+len1,y+len1-1);}
    if (x0<x+len1&&y0>=y+len1)  cover(x,y+len1,len1,x0,y0);
    else{board[x+len1-1][y+len1]=t;  cover(x,y+len1,len,x+len1-1,y+len1);}
    if (x0>=x+len1&&y0>=y+len1)  cover(x+len1,y+len1,len1,x0,y0);
    else{board[x+len1][y+len1]=t;   cover(x+len1,y+len1,len1,x+len1,y+len1);}
}
```

中国计算机学会
China Computer Federation

最近点对问题

- 给定平面上n(n>=2)个点,计算最近点对的距离。距离指欧几里得距离。可能有点重合，在这种情况下，它们之间的距离为0。这一问题可以应用于交通控制等系统中，在空中或海洋交通控制系统中，需要发现两个距离最近的交通工具，以便检测出可能发生的相撞事故。

最近点对问题

- 方法一：暴力枚举
- 共C(n,2)=n*(n-1)/2个点对，时间复杂度为O(n^2)。

最近点对问题

- 方法二：考虑用分治
- ans表示n个点的最近点对距离,先判重,如有点重合ans=0。
- 否则把n个点尽可能均分为左右两部分，ans的值只有以下3种情况：
- ①左边部分的最近点对距离d1
- ②右边部分的最近点对距离d2
- ③左半部分的点与右半部分点形成的最小距离d3
- ans=min(min(d1,d2),d3)
- 左右两部分的最近点对问题是与原问题结构性质一样的子问题，可以递归求解出d1,d2
- 第③部分d3的求解是问题的关键。

最近点对问题

- 分治法的时间复杂度$T(n)=2*T(n/2)+D(n)$，其中$D(n)$表示计算d3的时间复杂度。
- 如果d3的计算采用普通枚举法,两两枚举左右两部分的点对,则$D(n)=n*n/4$
- 设$n=2^k$
- $T(n)=2*T(n/2)+n*n/4$

  $=2*(2*T(n/4)+n*n/16)+n*n/4$

  $=4*T(n/4)+n*n/8+n*n/4$

  $=8*T(n/8)+n*n/16+n*n/8+n*n/4$

  $=\ldots=2^k*T(1)+n^2/4+n^2/8+\ldots+n^2/(2^(k+1))$

  $=(n^2+n)/2=O(n^2)$
- 并没有比暴力枚举快多少。

最近点对问题

- 如果D(n)=c*n，则T(n)=2*T(n/2)+c*n=O(nlogn)
- D(n)=c*n,说明计算d1,d2,d3时不能采用排序。做法如下：
- 1.**前期准备**：一开始，把输入的n个点复制到A和B数组中，对A数组按x坐标单调递增的顺序排序，对B数组按y坐标单调递增的顺序排序，并建立映射Index[],Index[i]表示B数组第i个点在A数组的第Index[i]位置,即B[i]与A[Index[i]]是同一个点。接下来进行判重，如有点重合，则返回答案为0。否则进入第二步开始递归
- 2.算法的每一次递归调用的输入都是点集对应的数组A和B,如果点数n<=3,则直接暴力枚举。如n>3则：

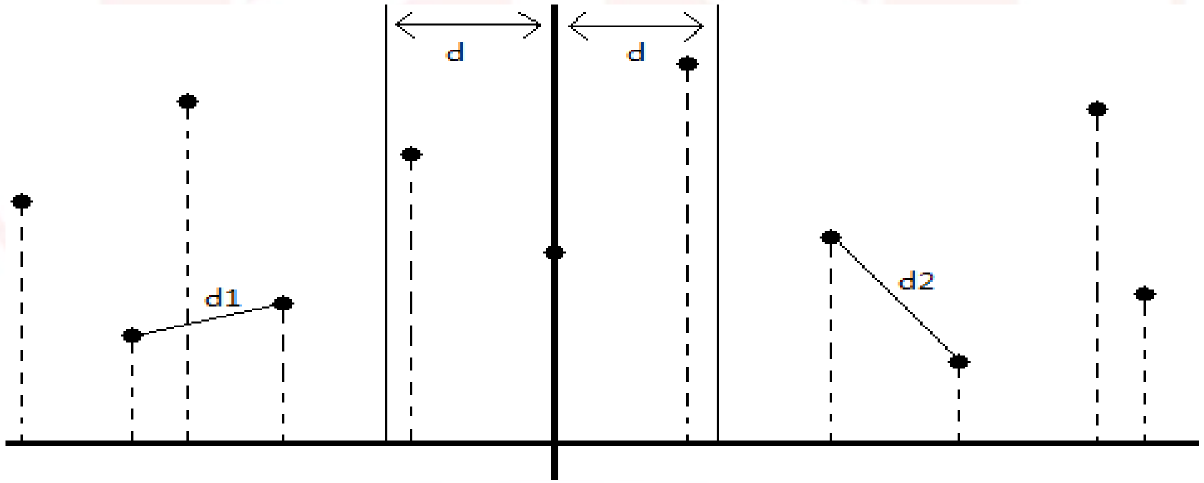
最近点对问题

- ①**划分问题**:直接用数组A的中间位置m作为分界点，A数组中m左侧(含m)的点为左半部分的点，m右侧的点为右半部分的点，数组A很容易就被划分为A_L和A_R，从左到右扫描B中的点,判断Index[i]与m的大小关系,如Index[i]<=m,则该点属于B_L否则属于B_R,同时更新B_L和B_R对应的Index[]。时间复杂度为O(n)
- ②**递归解决**：递归求解出左右两部分的最近点对距离d1,d2,取较小值d=min(d1,d2)

中国计算机学会 China Computer Federation

最近点对问题

- ③**合并问题：** 答案ans=min(d,d3),d3表示左半部分的点到右半部分点的最短距离，如果存在点对使得d3<d，则点对中的两个点必定都在距离直线L(分界线)d的单位之内。如图所示，它们必定处于以直线L为中心、宽度为2d的垂直带形的区域内。



- 为了找出这样的点对(如果存在的话)，算法需要做如下工作：

最近点对问题

- 1)是把数组B_L和B_R中所有不在宽度为2d的垂直带形区域内的点去掉后得到的数组,所以B_L和B_R也是按y坐标顺序排序的。
- 2)从上到下扫描B_L中的每一个点P，尝试找出B_R中与P距离小于d的点Q。通过分析将会看到，仅需考虑B_R中|Py-Qy|<=d的6个点,并记录下所有点对的最小距离d3。
- 3)如果d3<d则返回d3,否则返回d。
- 这样一来，合并问题的时间复杂度为O(n),总时间复杂度可以做到O(nlogn)。

CDQ分治

- CDQ分治是一种特殊的分治方法，在OI界初见于陈丹琦2008年的集训队作业中，因此被称为CDQ分治。
- CDQ分治通常用来解决一类“修改独立,允许离线”的数据结构题。实际上它的本质是按时间分治，即若要处理时间[L,R]上的修改与询问操作，就先处理[L,mid]上的修改对[mid+1,R]上的询问的影响，之后再递归处理[L,mid]与[mid+1,R],根据问题的不同，这几个步骤的顺序有时也会不一样。

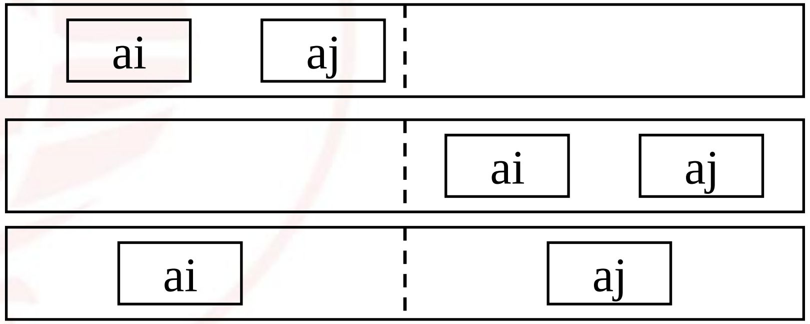
例题：逆序对

- 给定一个长度为N 的序列，定义它的逆序对数为二元组(i,j)，满足$i<j$ 且$A[i]>A[j]$。要求统计逆序对数。
- 例如：对于A=[3 2 5 1 4]，逆序对数为5：$3>2$, $3>1$, $2>1$,$5>1$, $5>4$。

中国计算机学会
China Computer Federation

例题：逆序对

- 下面介绍一种使用归并排序计算逆序对的方法(利用分治算法的思想)。
- 长度为N 的序列的逆序对数ans 可以分为3部分:
  - ① 左半段的逆序对数**L_ans**
  - ②右半段的逆序对数**R_ans**
  - ③**(i,j)**分立两侧的逆序对数**C_ans**



- 结论： ans = L_ans + R_ans + C_ans
- 例如：对于A=[3 2 5 1 4]，分成两段A1 = [3 2 5], A2 = [1 4]
- L_ans= 1(3>2), R_ans = 0, C_ans = 4(3>1, 2>1, 5>1, 5>4)

例题：逆序对

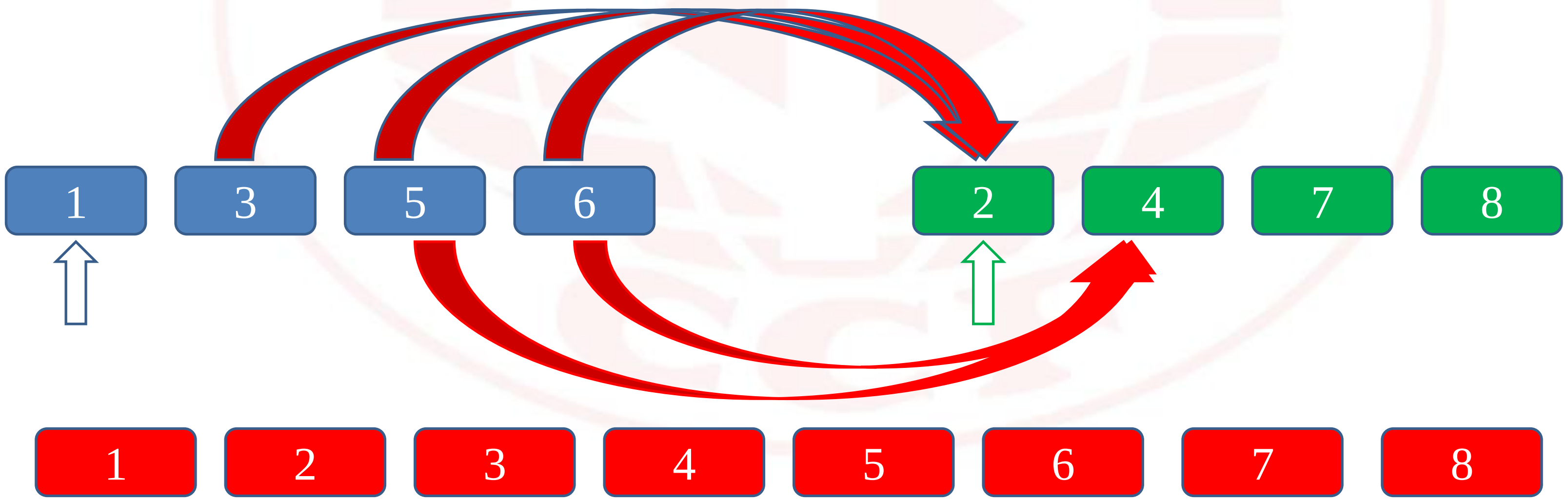
- ans = L_ans + R_ans + C_ans
- 其中L_ans，R_ans是与计算ans相同类型的子问题，递归计算即可。主要考虑(i,j)分立两侧的逆序对数C_ans:
- 结论：对两半段的数分别排序，(i 在左半段，j 在右半段，排在不同位置不影响先后关系)，C_ans 不变
- 例如：对于A=[3 2 5 1 4]，分成两段A1 = [3 2 5], A2 = [1 4]，对A1 和A2 进行排序，A’ = [2 3 5 1 4], C_ans 仍然为4(3>1,2>1, 5>1, 5>4)

例题：逆序对

- 考虑(i,j)分立两侧的逆序对数C_ans:
- 考虑在合并过程中，统计它们的分立两侧逆序对总数：
- 左部分区间为[L,m],右部分[m+1,r]，在合并过程中，左右指针分别为i,j，当出现A[i]>A[j]时,(i,j)是一个逆序对，且A[L..m]是升序,所以(i+1,j),(i+2,j)…(m,j)也都是逆序对，答案ans可以直接增加m-i+1。

例题：逆序对

- 如N=8，序列为[6, 1, 5, 3, 2, 4, 8, 7],分成[6,1,5,3],[2,4,8,7]左右两段, 递归计算出L_ans=4，R_ans=1,两部分排好序分别为[1,3,5,6],[2,4,7,8],C_ans的计算如下：



C_ans= 0 +3=3 +2=5

中国计算机学会
China Computer Federation

例题：3维偏序问题

- 有N个人，每个人有三种能力值A, B, C。第i个人的能力值为A_i, B_i, C_i。
- 如果$A_i>A_j$ && $B_i>B_j$ && $C_i>C_j$，称 i 比 j 有能力
- 现在要求出最长的一个序列$E=(E_1, E_2, \ldots, E_t)$，满足$E_i$比$E_{i-1}$有能力。
- 100%的数据：N<=40000。
- 为了简单起见，A, B, C都是1到n的排列。

例题：3维偏序问题

- 首先可以按照属性A从小到大把所有人排个序。
- 现在要求的是满足$j<i$, $B_j<B_i$, $C_j<C_i$的最长序列。
- 用F[i]表示以第i个人结尾的最长序列
- $F[i] = Max\{F[j] \mid j<i, B_j<B_i, C_j<C_i\} +1$
- 暴力处理，$O(n^2)$
- 若线段树套平衡树
- $O(n\log^2 n)$

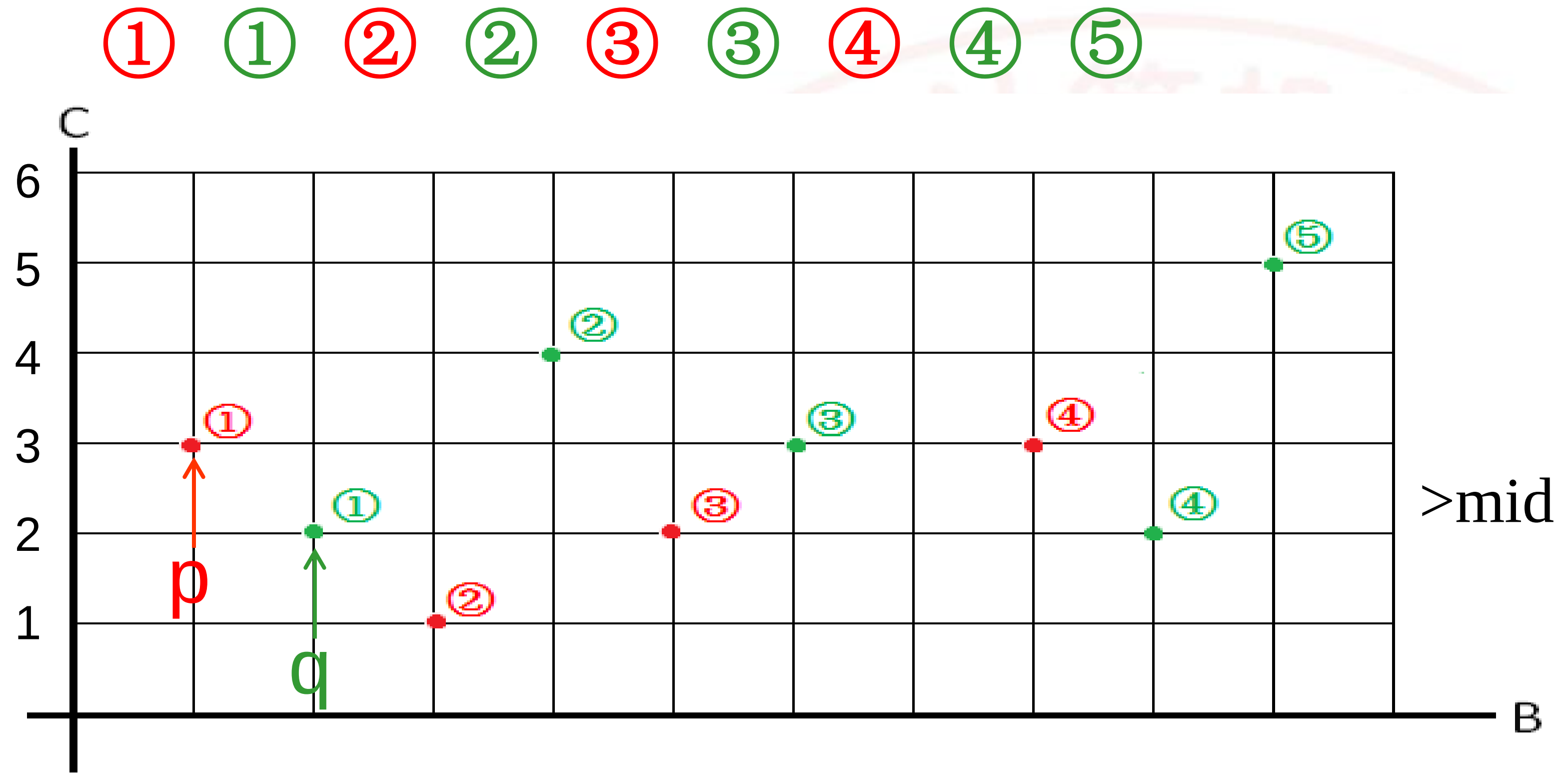
例题：3维偏序问题

- 尝试在这个问题上进行分治。
- 定义过程Solve(l,r),能够得到F[l]..F[r]的值,Solve(l,r)
  - **Solve(l,mid)**
  - **Solve(mid+1,r)**
  - 处理**[l,mid]**中元素对**[mid+1,r]**中**F[x]**取值的影响
- $F[x] = \mathrm{Max}\{F[j] \mid j<x,\ B_j<B_x,\ C_j<C_x\} + 1$
  - **1) x**在**[l, mid]**中：递归处理
  - **2) x**在**[mid+1,r]**中：递归解决
  - **3)**处理**[l,mid]**中元素对**[mid+1,r]**中**F[x]**取值的影响

例题：3维偏序问题

- 维护带权点集X=(B_i, C_i) (l≤ i ≤mid)，权值F[i]
- 支持询问：给定点(B_j, C_j) (mid+1≤ j ≤r)在点集X中寻找一个点(B_i,C_i)使得B_i<B_j且C_i<C_j，满足以上条件的点中取权值最大的。
- 离线处理
  - 将所有点和询问按Bi排序，按Bi顺序处理
  - 维护能够在一个位置插入数字和查询区间最大值的数据结构
  - 线段树或者树状数组

例题：3维偏序问题



update(3, 1)
F[q]+=sum(2)
update(1, 1)
F[q]+=sum(4)
update(2, 1)
F[q]+=sum(3)
update(3, 1)
F[q]+=sum(2)
F[q]+=sum(5)

红色表示的在[L, mid]中的点，绿色表示的在[mid+1,R]中的点，且在各自区间中B属性已经有序，处理完后[L, R]中的点B属性也有序。

例题：3维偏序问题

- 解法总结
  - 第一维：排序
  - 第二维：分治
  - 第三维：离线，数据结构
- 时间复杂度分析
  - T(n)=2T(n/2)+O(nlogn)
  - T(n)=O(nlog^2n)

China Computer Federation 中国计算机学会

例题：3维偏序问题

```
void CDQ(int L,int R)
{int mid=(L+R)>>1;
 if (L==R) return;
 CDQ(L,mid); CDQ(mid+1,R); //递归处理前半部分和后半部分
 int p=L,q=mid+1,tot=L;
 while (p<=mid && q<=R) //处理前半部分对后半部分的影响
   {if (e[p].B<=e[q].B) update(e[p].c, e[p].cnt), tmp[tot++]=e[p++];
    else e[q].f+=sum(e[q].c),tmp[tot++]=e[q++];
   }
 while (p<=mid) update(e[p].c, e[p].cnt),  tmp[tot++]=e[p++];
 while (q<=R) e[q].f+=sum(e[q].c),  tmp[tot++]=e[q++];
 for (int i=L; i<=mid; i++)  update(e[i].c, -e[i].cnt);  //为什么？
 for (int i=L; i<=R; i++)  e[i]=tmp[i];
}
```

中国计算机学会 China Computer Federation

例题：Cash(NOI2007)

- 有两种金券，金券按比例交易：买入时，将投入的资金，购买比例为Rate[i]的两种金券；卖出时，卖出持有的一定比例的金券。已知未来n天两种的金券价格A[i]、B[i]，初始资金为s，求最大获利。
- 提示：必然存在一种最优的买卖方案满足：每次买进操作使用完所有的人民币；每次卖出操作卖出所有的金券。
- 100%数据：1<=n<=100000。

例题：Cash(NOI2007）

- DP做法：
  - 令f[i]表示第i天能获得的最大收益。
  - 设x[i],y[i]表示第i天最多能持有多少A券,B券
- DP状态转移很明显：
  - x[i]=$\frac{Rate[j]*f[j]}{Rate[j]*A[j]+B[j]}$， y[i]=$\frac{f[j]}{Rate[j]*A[j]+B[j]}$
  - f[i]=$A[i]*x[j]+B[i]*y[j]$
  - 时间复杂度为O(N^2)
- 化简之：y[j]=$-\frac{A[i]}{B[i]}*x[j]+\frac{f[i]}{B[i]}$

例题：Cash(NOI2007)

- y[j]=$-\frac{A[i]}{B[i]} * x[j] + \frac{f[i]}{B[i]}$，令$k = -\frac{A[i]}{B[i]}$，y=y[j]，x=x[j]
- 在二维平面上定义点Xj=(x[j],y[j])
- 维护一个点集X，支持以下两个操作：
  - 1)在第一象限的任意位置插入一个点(x[i],y[i])
  - 2)给定负数斜率k，求所有斜率为k且过点集X中任意点的直线在Y轴上的最大截距
- 操作2最终用到的点都会在点集的上凸壳上
  - 维护点集X的凸包，支持动态插入和斜率查询
  - 平衡树结构 或 set 维护，时间复杂度为O(nlogn)

例题：Cash(NOI2007)

- 算法存在的问题
  - 不能用斜率优化DP，问题在于这个式子的x，也就是x[j]是并不单调递增的
  - 边界情况众多，难写难调

例题：Cash(NOI2007)

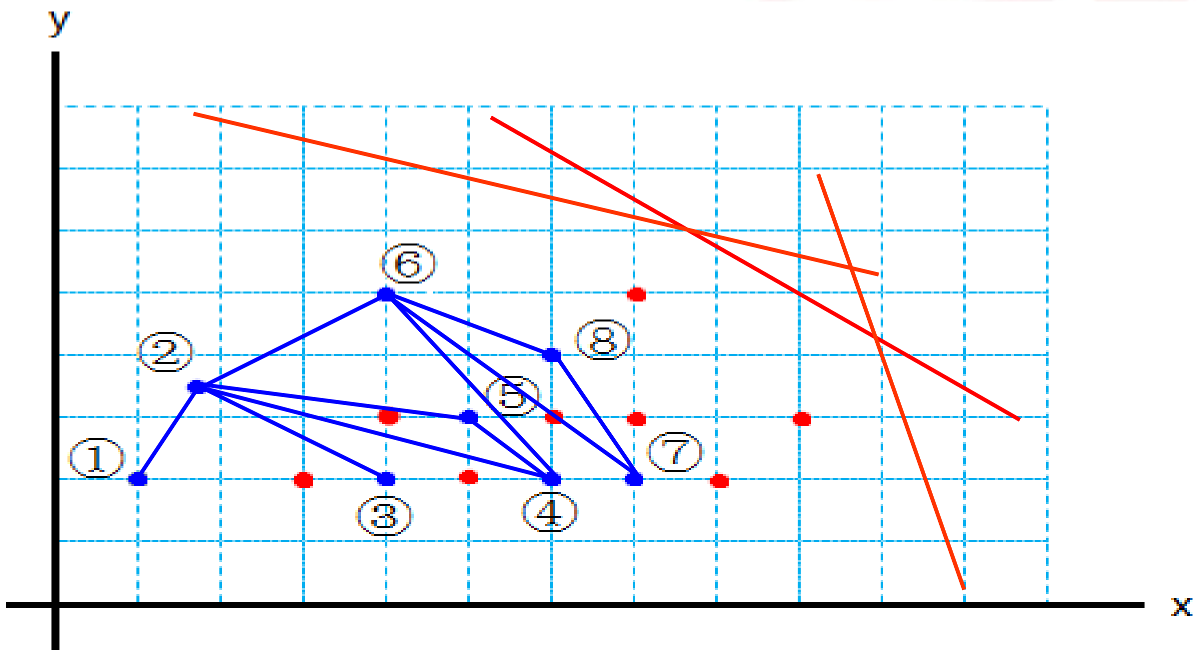
- 我们来考虑分治，定义过程Solve(L,R)
- 假设运行Solve(L,R)可以得到F[L]到F[R]的值。
  - [L,mid]区间里的询问,可以直接递归Solve(L,mid)解决。
  - [mid+1,R]区间里的询问k，会受到[mid+1,k]这些点的影响，以及[L,mid]的影响。前半部分可以递归解决。
- Solve(L,R)
  - 递归调用Solve(L,mid)
  - 整体考虑[L,mid]间的点对[mid+1,R]间询问的影响。
  - 递归调用Solve(mid+1,R)

例题：Cash(NOI2007)

- 整体考虑[L,mid]对[mid+1,R]的影响
- 给定点集X和一系列询问
  - 每个询问是一个负数斜率
  - 回答所有斜率符合且通过点集X中任意点的直线中，Y轴的最大截距是多少
- 只要考虑点集X的上凸包。
  - 对于每个询问，在凸包上二分即可。
- 这一步的复杂度是O(nlogn)，这里n=r-l。

中国计算机学会 China Computer Federation

例题：Cash(NOI2007)



1、求[L, mid]中点集的上凸壳，如果采用二分法求动态凸壳，则时间O(nlogn)

2、对于[mid+1,R]中的一个询问是一个负数斜率k，二分找到斜率为k的直线扫到的点，与前驱斜率k_1，与后继斜率k_2满足$k_1 \geq k \geq k_2$

蓝色的点表示[L,mid]中的点，红色的点表示[mid+1,R]中的点。圆圈内的数字表示点的编号(即第几天对应的点)

例题：Cash(NOI2007)

- 时间复杂度？
- Solve(l,r)的复杂度是O(nlogn)
- T(n)=2T(n/2)+O(nlogn)
- T(n)=O(nlog^2n)
- 离最优化还有距离。
- 需要log n的地方
  - 1) 求点集凸包
  - 2) 二分答案

例题：Cash(NOI2007)

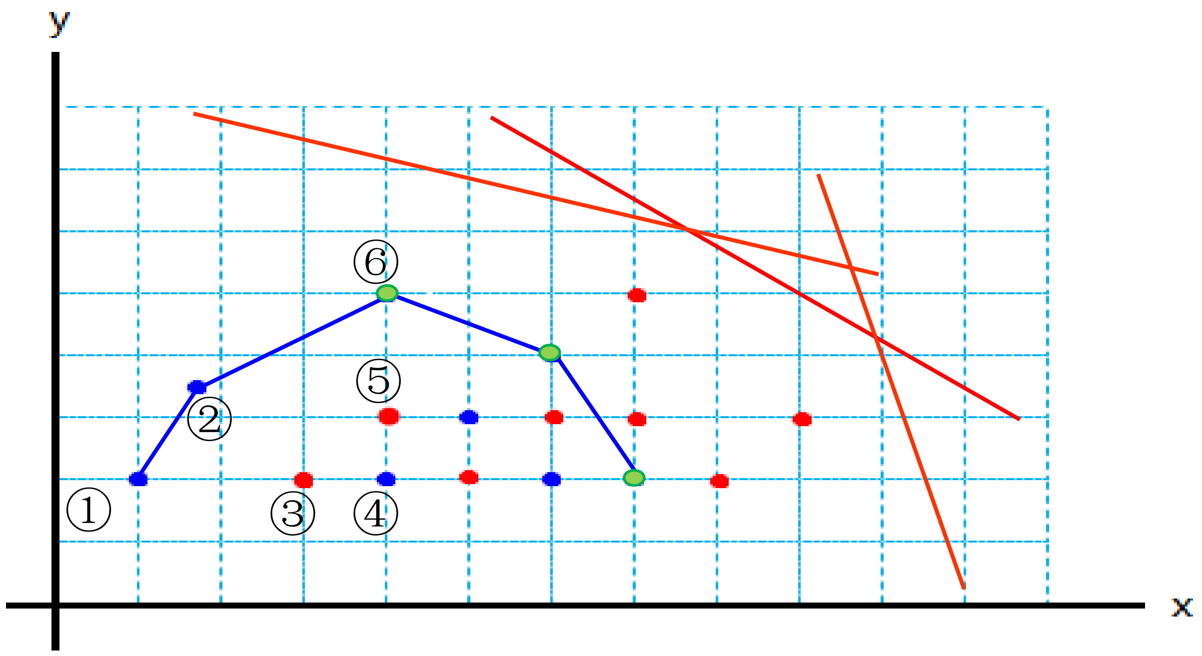
- 1) 点集凸壳
  - 求凸壳采用Graham扫描法，时间复杂度是O(n)
  - 求凸包时间复杂度是O(n)基于点集已经按先x后y排好序
  - Solve(L,R)结束后返回[X_L..X_R]的点集有序(先x递增,x相同y递增)
- 2) 二分答案
  - 放弃二分查找，离线处理
  - 把点集凸壳和所有询问排序，用两个指针扫描
- 3) 对询问排序
  - 每个询问需额外记录两个信息：位置pos和斜率k
  - 提前对询问进行一次排序，保证分治的每一块斜率是有序的
  - 预处理复杂度O(nlogn)，主递归中单步O(n)
- T(n)=2T(n/2)+O(n),T(n)=O(nlogn)

例题：Cash(NOI2007)

- 如何保证分治的每一块斜率是有序的？
  - 每个询问记录信息：位置pos和斜率k、A、B、Rate等
  - 在调用solve(1,n)前将所有询问按斜率从小到大排序
  - 在solve(L,R)过程的一开始就对询问分组，询问位置pos≤mid的分成一组，pos>mid的分成一组，每一组内是按斜率从小到大排序的。

```
//对询问集合排序，先位置后斜率
int mid=(L+R)>>1, l1=L, l2=mid+1;
for (int i=L; i<=R; i++)
   if (q[i].pos<=mid)  nq[l1++]=q[i];
   else nq[l2++]=q[i];
for (int i=L; i<=R; i++) q[i]=nq[i];
```

例题：Cash(NOI2007)



蓝色的点表示[L,mid]中的点，
红色的点表示[mid+1,R]中的点。

1、求[L, mid]中点集的上凸壳，采用Graham扫描法，则时间复杂度为O(n)

2、因为询问的斜率已经有序，对于[mid+1,R]中的一个询问是一个负数斜率k，在刚才找到点的基础上继续找斜率为k的直线扫到的点，则时间复杂度为O(n)

3、将[L, R]中点集的点先x递增,x相同y递增进行归并排序，则时间复杂度为O(n)

```
void solve(int L,int R)
{if (L==R) //此时L之前包括L的f值已经达到最优,计算出对应的点即可
  {f[L]=max(f[L-1],f[L]);
   p[L].y=f[L]/(q[L].a*q[L].rate+q[L].b);    p[L].x=p[L].y*q[L].rate;    return ;
  }
 int mid=(L+R)>>1, l1=L, l2=mid+1;
 for (int i=L; i<=R; i++) //对询问分组
   if (q[i].pos<=mid) nq[l1++] = q[i]; else nq[l2++] = q[i];
 for (int i=L; i<=R; i++) q[i] = nq[i];
 solve(L, mid); //递归左区间
 int top=0;
 for (int i=L; i<=mid; i++) //左半区所有点都以计算好把它们入栈维护上凸壳
   {while (top>=2 && slope(i,st[top])+eps > slope(st[top],st[top-1])) top--;
    st[++top]=i;
   }
 for (int i=R,j=1; i>=mid+1; i--) //拿左半区更新右半区保证询问斜率递减
   {while (j<top && q[i].k < slope(st[j],st[j+1])+eps) j++;
    f[q[i].pos]=max(f[q[i].pos],  p[st[j]].x*q[i].a + p[st[j]].y*q[i].b);
   }
 solve(mid+1, R);  //递归右区间
 l1=L, l2=mid+1;  //合并左右区间的点,按照x,y排序
 for (int i=L; i<=R; i++)
   if ((p[l1]<p[l2] || l2>R) && l1<=mid) np[i] = p[l1++];
   else np[i] = p[l2++];
 for (int i=L; i<=R; i++) p[i]=np[i];
}
```

整体二分

- 二分答案可以说是整体二分的前世。
- 整体二分类似于一些决策单调性的分治，可以解决诸多区间第k小或区间第k大的问题。
- 整体二分中需要实现一个重要函数solve(l,r,L,R)表示表示询问编号在l~r的操作的答案在L~R这个区间，具体看下面静态区间第k小这个例子。

例题：K-th Number

- 给定一个N（N≤10^5）个元素的序列A，元素|Ai|≤10^9，有Q（Q≤5×10^3）次查询，形如x, y, k：查询[x, y]的第k小的数。

例题：K-th Number

- 这是一道可持久化线段树的模板题。
- 重点介绍整体二分的做法。先看两个例子：
- 例子1：
  - 给定一个正整数序列A及固定的整数S，执行M次操作
  - 每次查询l~r间不大于S的数的个数
  - 用树状数组维护一下……
- 例子2：
  - 给定一个正整数序列，求此序列的第K小数是多少。
  - 二分答案……

例题：K-th Number

- 将输入a[i]=x也看成询问，记录询问如下信息：编号id、左端点x、右端点y、查询k(第k小)、类型type(输入还是询问)。
  - 输入a[i]: id=i, x=a[i], y=1, k=0, type=1
  - 询问x[i],y[i],k[i]: id=i, x=x[i], y=y[i], k=k[i], type=2
- 整体二分
  - Solve(l,r,L,R)表示询问编号在[l,r]的答案在[L,R]范围内
  - 边界：L==R，编号在[l,r]的答案都等于L
  - 对答案[L,R]进行二分mid=(L+R)/2
  - 对询问编号在[l,r]进行分组lq，rq，lq的答案都是[L,mid]范围内, rq的答案都在[mid,R]范围内。

例题：K-th Number

- 如何分组
  - Type=1，如果x<=mid则放入lq否则放入rq
  - Type=2，记区间[x, y]中≤mid的数有cnt个然后将这些询问分类:
  - 1.若k≤cnt则说明第i个询问的答案在[L, mid]中
  - 2.若k>cnt，则说明第i个询问的答案在[mid+1, R]中，且等价于在值域[mid+1, R]中查询第(k−cnt)小的数
  - 对于统计cnt可以利用树状数组维护
- 然后分别把上面两类分开递归处理即可
  - Solve(l, l+|lq|-1, L, mid), |lq|表示lq中询问的个数
  - Solve(l+|lq|, r, mid+1, R)

```
void solve(int l,int r,int L,int R) //询问编号在[l,r]的答案在[L,R]
{if (l>r) return;
 if (L==R)  //答案唯一了
   {for (int i=l; i<=r; i++)   if (q[i].type==2)  ans[q[i].id]=L;    return; }
 int mid=(L+R)>>1, lnow=0, rnow=0;
 for (int i=l; i<=r; i++)  //处理每一个询问进行分组
   {iT(q[i].type==1)  //是询问类型1
       {if (q[i].x<=mid)  {add(q[i].id,q[i].y); lq[++lnow]=q[i];  }
         else  rq[++rnow]=q[i];
       }
    else //是询问类型2
       {int cnt=sum(q[i].y)-sum(q[i].x-1);
         if (q[i].k<=cnt)  lq[++lnow]=q[i];   //答案在[L,mid]
         else {q[i].k-=cnt;  rq[++rnow]=q[i]; }  //答案在[mid+1,R]
       }
   }
 for (int i=1; i<=lnow; i++)  //撤销在树状数组的影响
   if (lq[i].type==1) add(lq[i].id, -lq[i].y);
 for (int i=1; i<=lnow; i++)  q[l+i-1]=lq[i];
 for(int i=1;i<=rnow;i++)   q[l+lnow+i-1]=rq[i];
 solve(l,l+lnow-1,L,mid); //递归处理
 solve(l+lnow,r,mid+1,R);
}
```

例题： Dynamic Rankings

- 给定一个含有 n 个数的序列 $a1,a2\ldots an$，需要支持以下两种操作，操作总数为m：
- Q l r k 表示查询下标在区间$[l,r]$ 中的第k 小的数
- C x y 表示将 ax 改为 y
- 100% 的数据，$1\le n,m\le 10^5$，$1\le l\le r\le n$，$1\le k\le r-l+1$，$1\le x\le n$，$0\le ai,y\le 10^9$。

分析

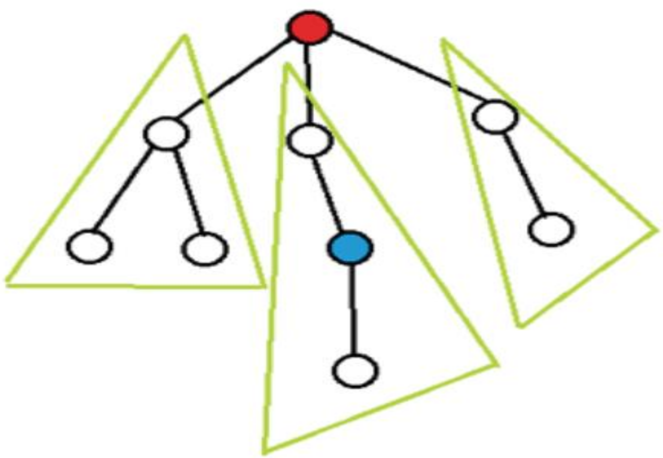
- 其实动态区间第k小和静态区间第k小没有什么差别，我们将修改操作分解成两个部分放入操作序列即可：一次删除和一次添加
- 然后照上面那个题做法， 在扫到权值的时候判断是删除还是添加并判断是否≤mid就好了。

点分治

- 经典问题：给定一棵含有n个顶点的树，第i条边连接着顶点u_i与v_i，它的长度为w_i。现在要求统计，最短距离不超过k顶点的对数。
- 上面所提到的问题求的是顶点对数，而实际上一对顶点就代表着树中的一条路径。点分治是一种用来解决树的路径问题的高效算法，在竞赛中有着较为广泛的应用。

点分治

- 基于点分治，即选择树上一点为根，将无根树变成有根树，对其子树之间的信息进行统计及运算，再对它的每个子树递归操作。
- 注意到这样得到的实际操作树和原树同构的概率很低。



点分治

- 对于点分治，可以证明对于结点个数为N的树，我们一定能够找到一点root，使得以该点为根，其子树大小均不超过N / 2。
  - 假设某一子树u大小为cnt(u)，且cnt(u)超过N / 2，我们就把root移到u的位置，由于N - cnt(u) < N / 2，也即从移动方向而来的子树一定不会不合法，故我们不断移动一定能找到一个满足条件的root。
  - height(N) ≤ height(N / 2) + 1，也即height(N) ≤ logN。

点分治

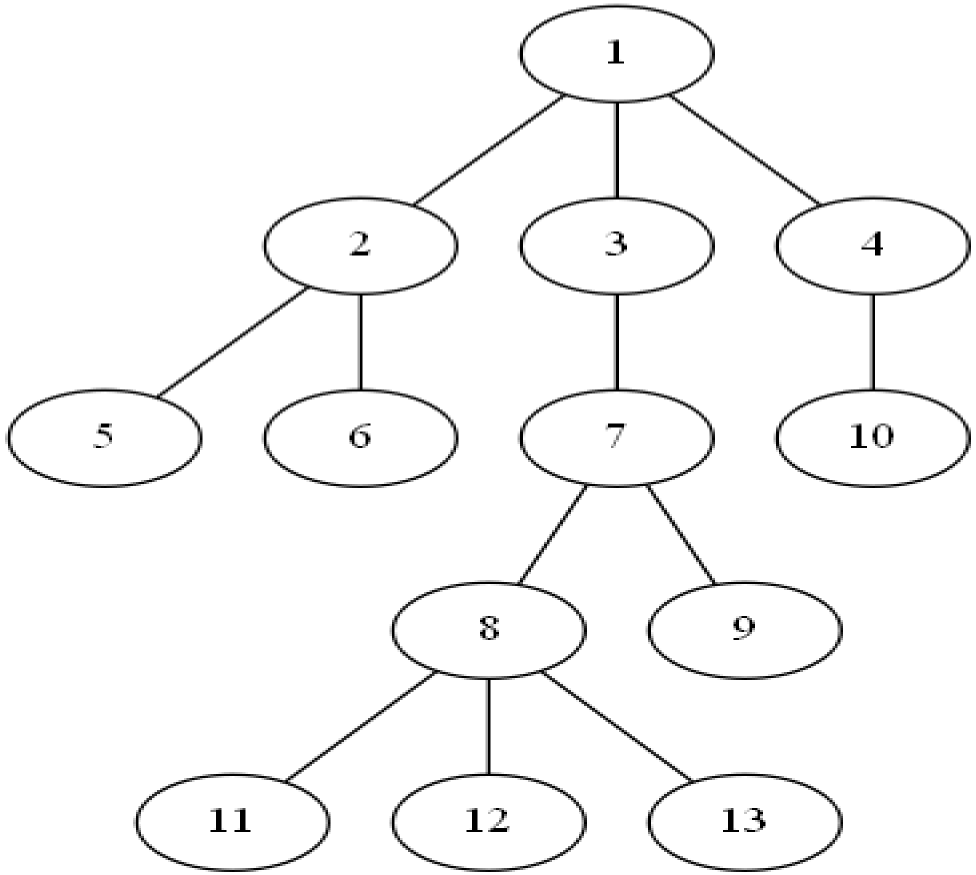
- 点分治的思路是非常简单的：
  - 在一个给定的树中找到一点，使得它最大的子树大小最小，这一点是当前树的重心，作为我们此次实际操作树的根。
  - 遍历当前树的每一个结点，统计并计算子树间对答案的贡献。
  - 在原树中删除这个重心，对其子树递归。

树的重心

- 一棵树的重心定义为一棵树的一个节点，使得删去这个节点后，节点最多的连通块的结点个数最小，即不会超过整棵树大小的一半。
- 重心可以通过树上的动态规划来解决：
  - DFS一次算出以每个点为根的子树的大小;
  - 选一个点u为根并删去后，结点最多的联通块的结点个数为 **max{size[son[u]],n-size[u]}**
  - 从中选取最优的那个点作为这一次的根
- 这样做一次的时间复杂度为O(n)

树的重心

- 如图，3是这棵树的重心，因为删去3后原树被分成了2个大小为6的连通块。
- 而1不是这棵树的重心，因为删除1后原树最大的连通块大小为7。



树的重心

```
void dfs(int x,int fa) //求以x为根的每个节点的子树大小
{sz[x]=1;
 for(int i=h[x]; i; i=nxt[i])
   if (!del[to[i]] && to[i]!=fa) //del[i]经过该点的路径是否处理完
        dfs(to[i],x),sz[x]+=sz[to[i]];
}
int getweight(int x) //求以x为根的子树的重心
{int i,now,t=x;
 dfs(x,0);
 while(1)
   {now=t, t=0;
    for(i=h[now]; i; i=nxt[i])
        iT(sz[to[i]]>sz[t] && sz[to[i]]<sz[now] && !del[to[i]]) t=to[i];  //找到最大的子树
    if (sz[t]<=sz[x]/2) break; //满足重心的条件
   }
 return now;
}
```

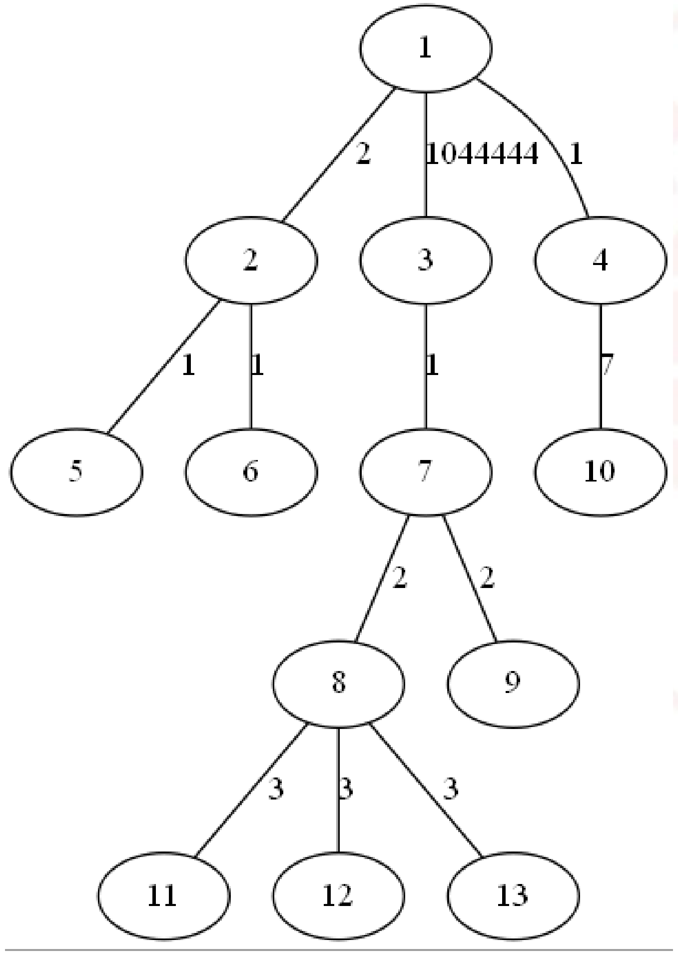
点分治

```
void conquer(int u) //处理u所在的联通块
{u=getweight(u); //找出该联通块的重心
 work(u); //处理该联通块中通过点u的所有路径
 del[u]=true; //将点u删除
 for(int i=h[u]; i; i=nxt[i]) //枚举每一个与u相邻的点
   if (!del[to[i]]) conquer(to[i]); //如果未被删除则递归处理
}
```

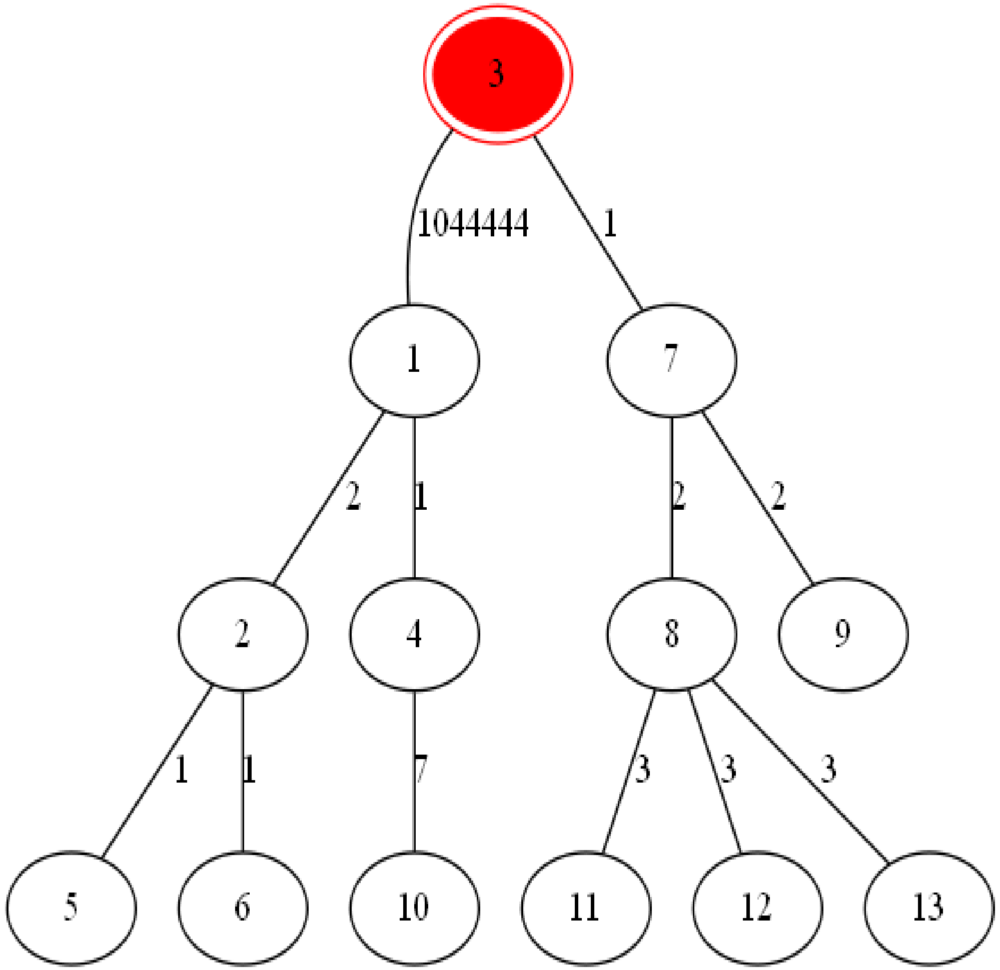
例题：树上的询问

- 一棵n个点的带权无根树，有p个询问，每次询问树中是否存在一条长度为Len的路径，如果是，输出Yes，否输出No。
- 100%的数据：n≤10^4，p≤100，Len≤10^6。

点分治

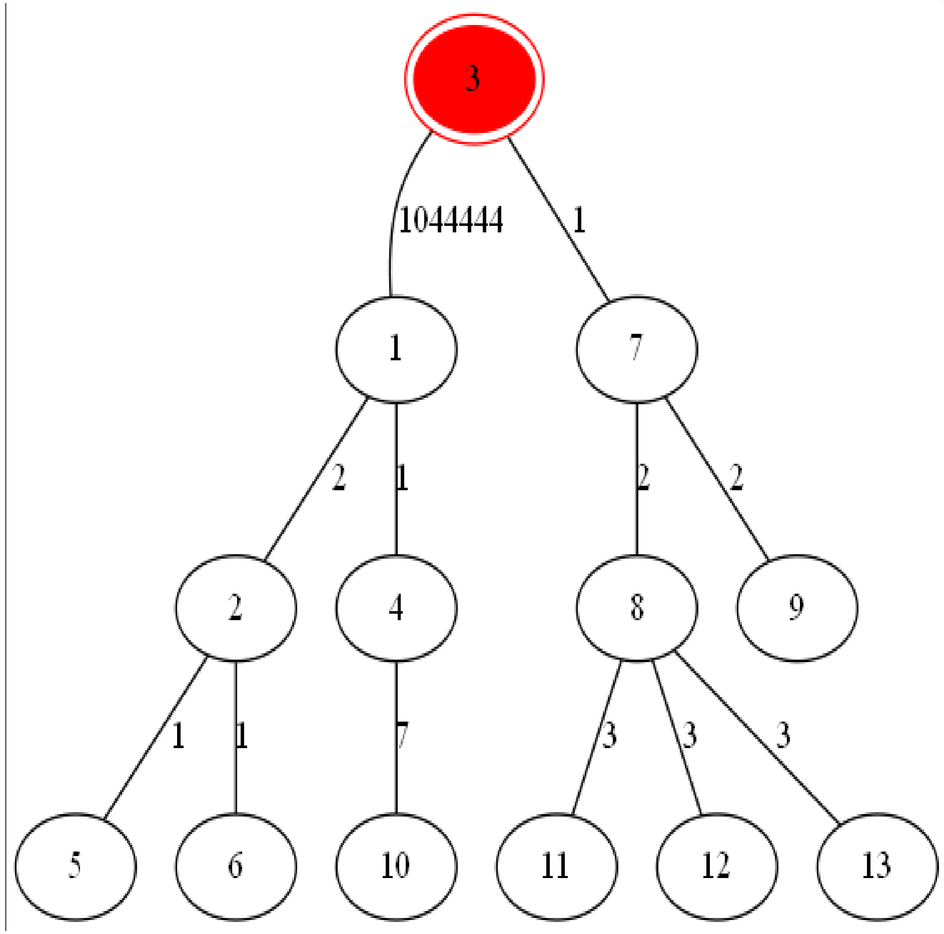


1.以一定的方式选取一点作为根，将无根树转为有根树。



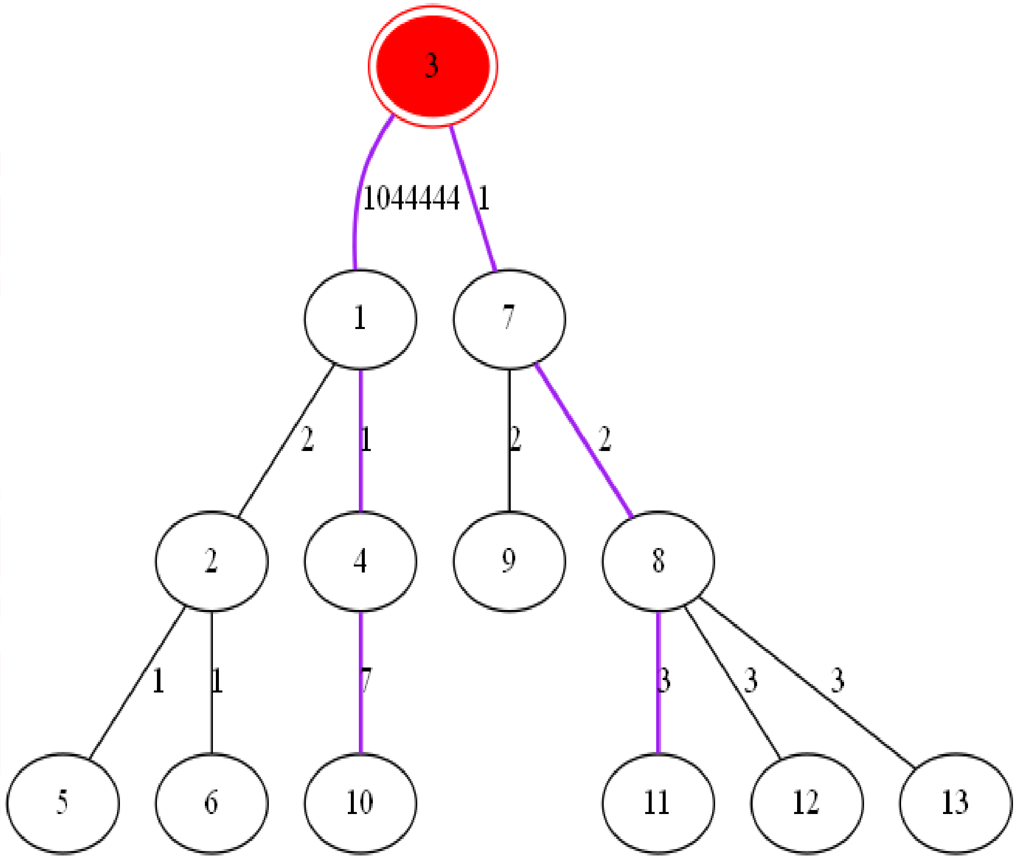
中国计算机学会
China Computer Federation

点分治

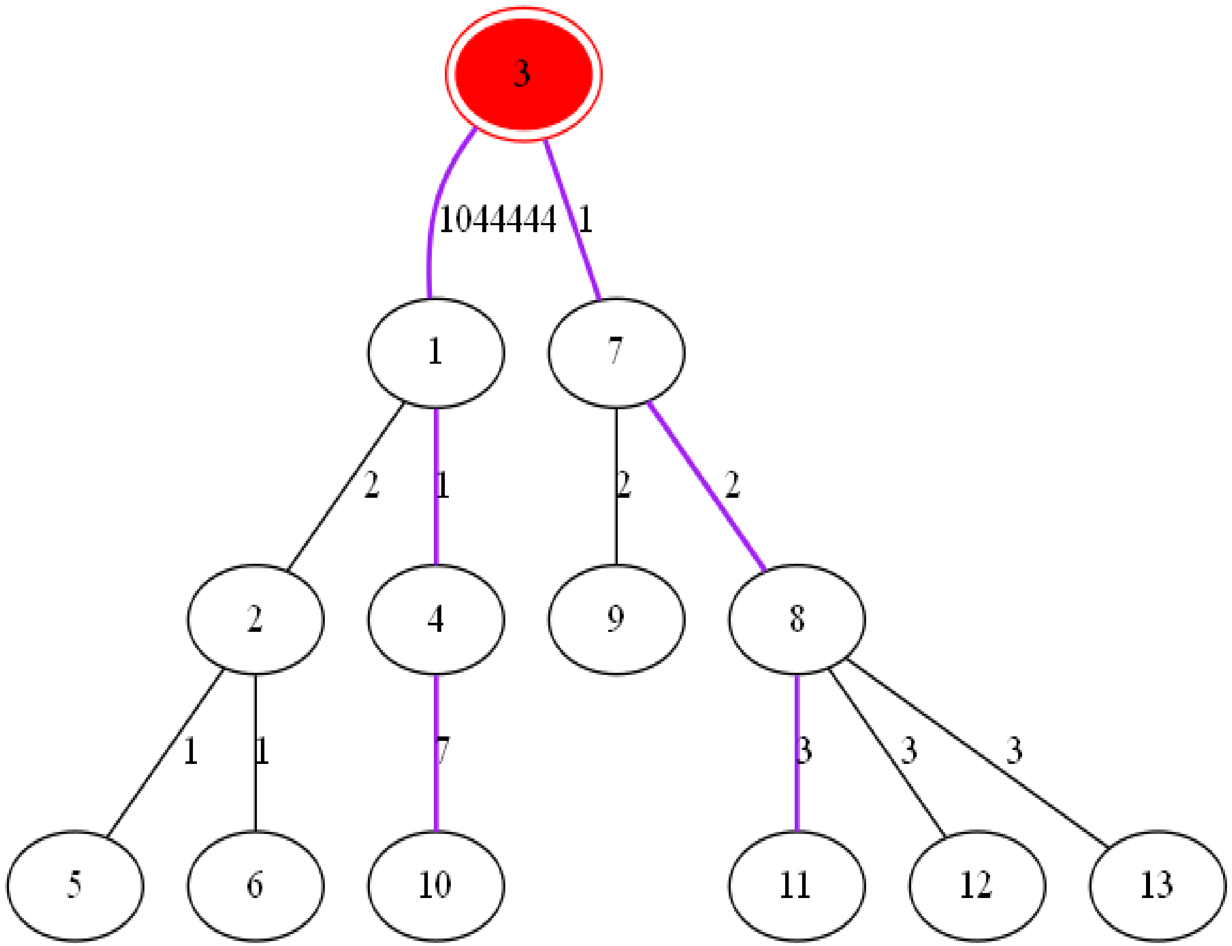


2.处理所有经过选取的这点(如图中3号节点)的路径对答案的影响。

例如查询图中是否有长度为1044458的路径。

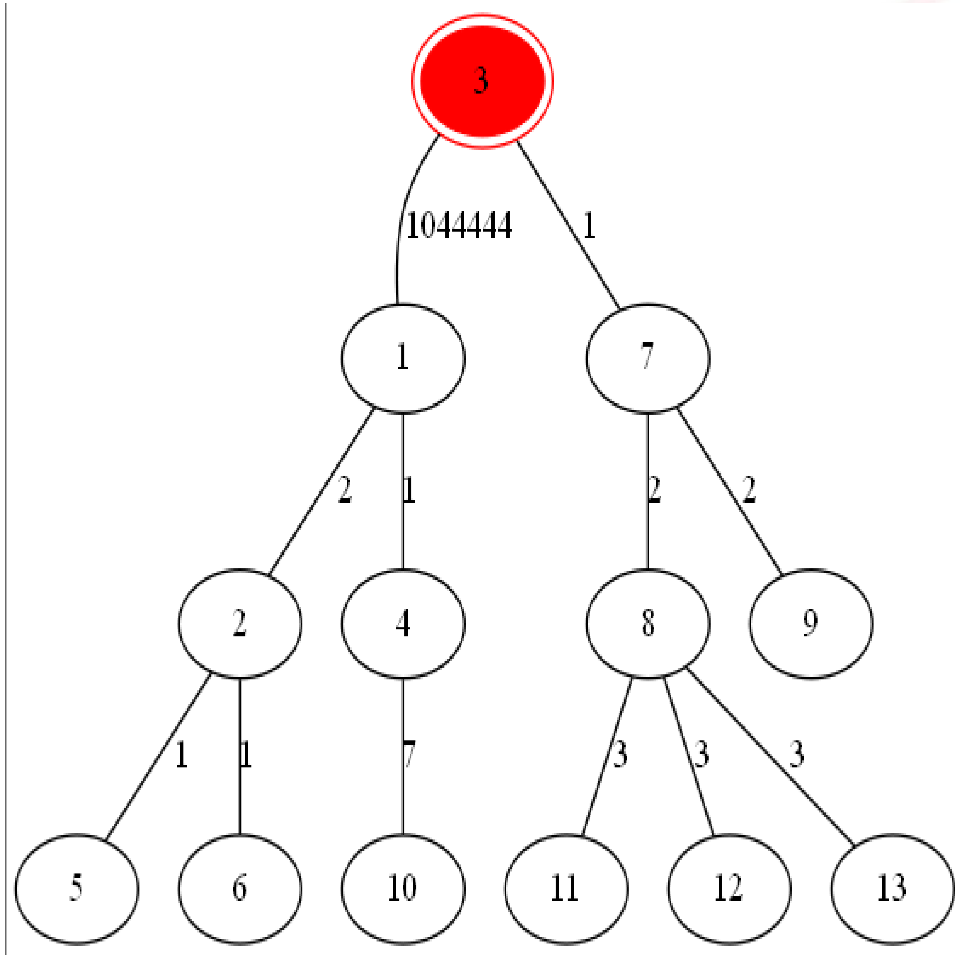


点分治

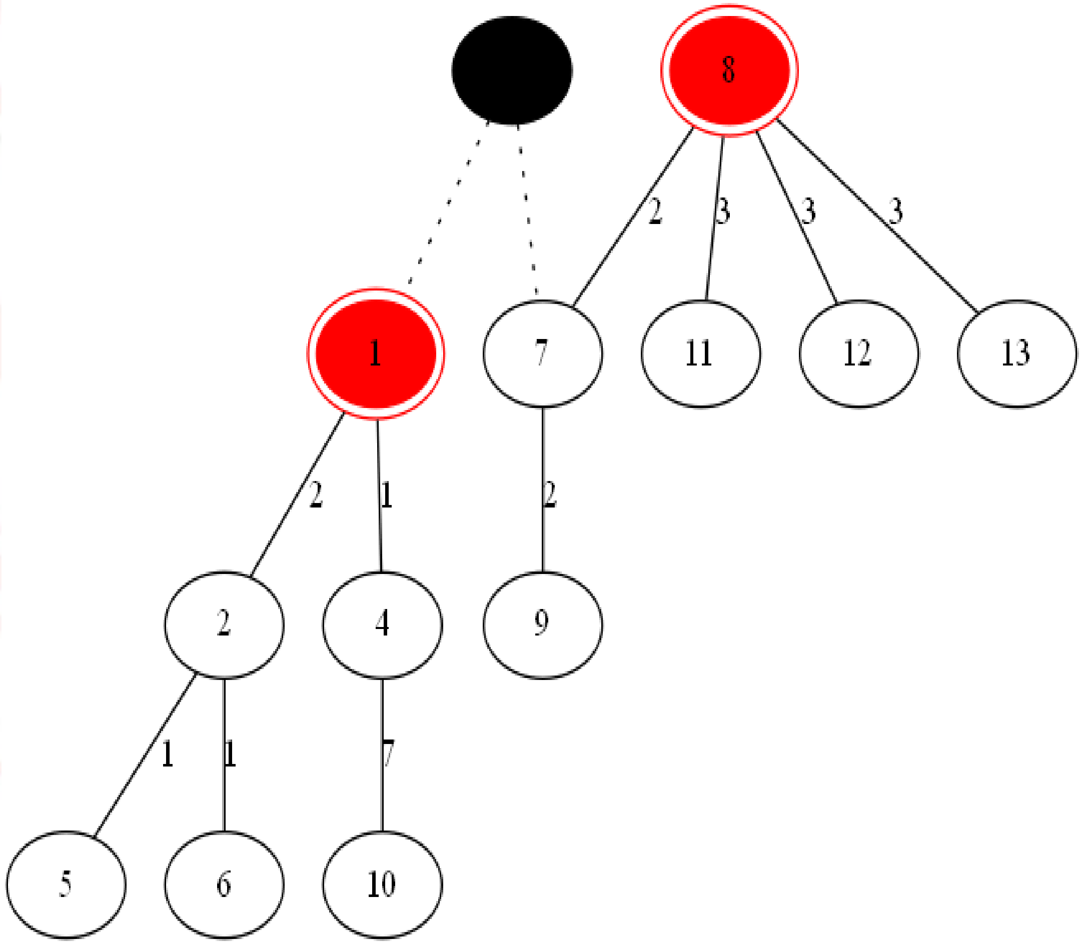


- 那么既然我们查询到了长度为1044458的路径，我们能否查询到长度为2的路径呢？
- 由于没有长度为2的路径**经过根**，我们在这一步查询不到长度为2的路径。

点分治



3.**删除**上一次选取的节点，对每个子树**重复操作1,2**，直到不再有子树。



分析

- 在每一棵点分治的树中只考虑经过根的路径
- （1）某一点到根的路径
  - 只需要算出每个点到根的距离即可判断
- （2）来自根节点不同儿子所在子树的两个点构成的路径
  - 每个点要记信息： bel[i], dis[i]分别表示点i在删除根后属于哪个连通块，到根的路径长度
  - 即求是否存在j，使得bel[i]≠bel[j]且dis[i]+dis[j]=Len 。

分析

- 在实际处理时可以：
  - 依次处理根的每一棵子树；
  - Hash[i]表示已经处理的子树中到根距离为i的点是否存在，特别注意Hash[0]=true，为了计算某一点到根的路径长度为Len的路径；
  - 当处理完一个子树时，子树中每个点i到根的路径长度dis[i]都已经知道，对于询问Len，只需判断Hash[Len-dis[i]]是否为true;
  - 再用刚刚计算子树的dis[]来更新Hash[ ]。

例题：IOI2011 Race

- 给一棵N个节点的树，每条边有权值；
- 求一条简单路径，权值和等于K，且边的数量最小。
- 数据范围：$N \le 200000$，$K \le 1000000$。

分析

- 在每一棵点分治的树中只考虑经过根的路径
- （1）某一点到根的路径
  - 需要算出每个点到根的距离和经过的边数
- （2）来自根节点不同儿子所在子树的两个点构成的路径
  - 每个点要记三个信息：bel[i], dis[i], cnt[i]分别表示每个点在删除根后属于哪个连通块，到根的路径长度，到根的路径上经过的点数;
  - 即求min{cnt[i]+cnt[j] | bel[i]≠bel[j]且dis[i]+dis[j]=K

分析

- 在实际处理时可以：
  - f[i]表示已经处理的子树中到根距离为i的点中cnt最小为多少；
  - 依次处理根的每一棵子树得到dis[ ]和cnt[ ];
  - 当处理完一棵子树，对子树中点j所能匹配的点到根的距离都是固定的，直接拿出对应的f值更新答案即可，即ans=min{ans,cnt[j]+f[K-dis[j]]}
  - 用这一棵子树的dis[]和cnt[]更新f[ ]，即f[dis[j]]=min{f[dis[j]], cnt[j]}。

分析

- 有几点优化很重要：
  - f[ ]初始化，f[0]=0, f[i]=n (i=1...K)i只需到K，无需把整个f[ ]初始化；
  - 在DFS计算一棵子树中所有点j到根（即当前重心）的dis[j]和cnt[j]时有一个强有力的剪枝：dis[j]>K或cnt[j]>=ans时则return;
  - 当换一个重心为根的时，f[ ]需要还原，这时若操作f[i]=n (i=1...K)肯定超时，优化的方法是把这个重心为根时算到的dis用一个数组记下，只还原记下的dis值的f[ ]的元素。

例题：重建计划

- 给出一棵树，有边权，找出其中一条包含了不少于L，不多于R条边的路径，使得其边权的平均值最大。
- $n \le 50000$，边权 $\le 1e7$。

分析

- 题目让我们求的是式子$\max\left\{\frac{\sum_{<i,j>\in S} v_{<i,j>}}{|S|}\right\}$，S是一条树上的路径，且L≤ |S| ≤R。
- 显然我们二分一下这个值，设其为mid，再判断是否有解，即可转化为是否存在$\sum_{<i,j>\in S}(v_{<i,j>} - mid) \geq 0$的树上路径，且L≤ |S| ≤R。

分析

- 求树上的路径可以用树分治来做了，分治到以某个节点u为重心时，就可以得到经过当前重心u的所有路径答案。
- 具体方法是：
  - 计算重心u的某一棵子树时，递归算出每一个节点v到u的深度deep(即边的条数)以及u到v的路径价值和dis，同时更新深度为deep时最大的路径价值和 $cur[deep] = max\{cur[deep], dis\}$。

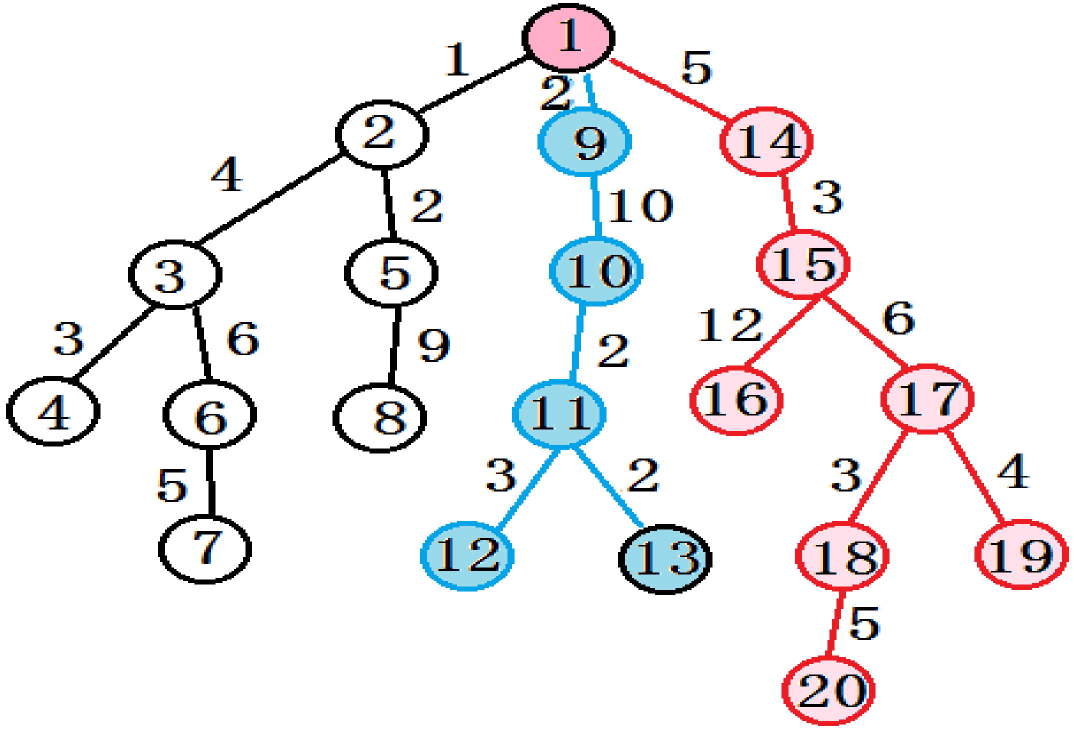
分析

- 具体方法是：
  - 以前已经处理过的u的子树中，lst[i]记录深度为i时的最大路径价值和。
  - 在当前处理的子树中枚举深度(边数)为j，贪心应在以前已处理过的子树中找深度在[L-j, R-j]范围内最大路径价值和，即$\max\{lst[i] \mid L-j \le i \le R-j\}$。
  - 用当前子树的cur[ ]更新lst[ ]，清空cur[ ]，为处理下一棵子树准备。
- 暴力做时间复杂度为$O(n^2 log^2 n)$，会TLE。

分析

- 如何优化呢？针对第三步的暴力进行优化
  - 从大到小枚举路径的一端在当前处理的子树中的深度(边数)j，路径的另一端在以前已处理过的子树中的深度范围在[L-j, R-j]内，显然这个区间是往后移动的，所以可用单调队列维护lst[ ]的下标。
  - 单调队列中维护的lst[ ]的下标(即深度)是递增的，但lst值确实单调递减的。
  - 当然单调队列改成线段树也可以。
- 二分+分治，复杂度是$O(nlog^2n)$的。

分析



1号点是重心，蓝色和红色的子树已处理过，黑色子树当前正在处理。

- cur[1]=1, cur[2]=5, cur[3]=12, cur[4]=16
- lst[0]=0, lst[1]=5, lst[2]=12, lst[3]=20, lst[4]=18, lst[5]=22
- 例如L=4, R=6时，

  j=4时，单调队列[2],

  j=3时，单调队列[3],

  j=2时，单调队列[3 4],

  j=1时，单调队列[5]。

Thanks