算法分析与设计 II

2022-2023-2

数学与计算机学院
数据科学与大数据技术

Last Modified: 2023.1.16

6. 高级数据结构

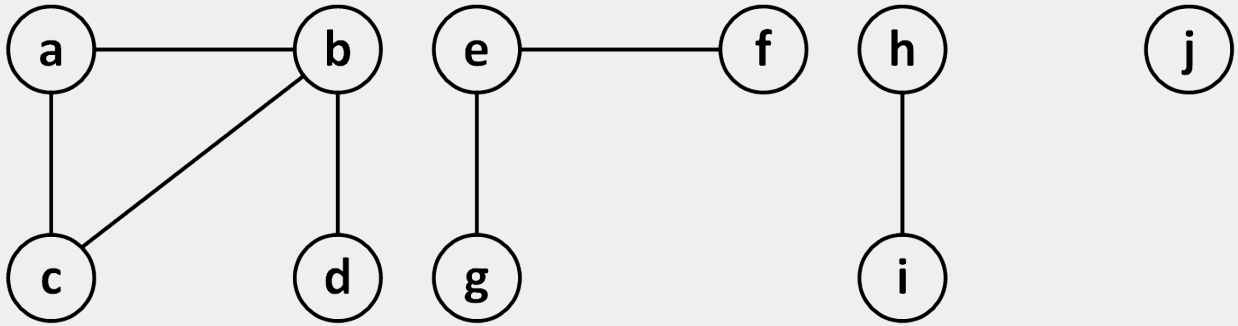
6.1 并查集

- 不相交集合数据结构(Disjoint Set)：将编号分别为 1...*n* 的 *n* 个对象划分为不相交集合，在每个集合中，选择其中某个元素 *x* 代表所在集合
- 基本操作：
  - MAKE-SET(x) 建立新的集合，唯一成员 *x*
  - UNION(x,y) 将包含 *x* 和 *y* 的两个集合合并成一个新的集合
  - FIND(x) 返回指针，指向包含 *x* 的集合的“代表”(representative)
- 由于两个基本操作是UNION(合并) 和FIND(查找)，所以称为“并查集”

实现

- 使用树的数据结构来表示并查集，在程序实现起来相对简单，森林中的每棵树作为一个集合，树的节点表示集合中的元素，树的根用来作为该集合的“代表”
  - 查找操作相当于对树进行遍历，从某个节点沿着父节点指针找到这棵树的根节点
  - 合并操作相当于两棵树合并成一棵树，两个节点位于两棵不同的树的时候，将一个节点所在树的根的父亲节点指向另一个节点所在树的根
- 在具体程序实现中，使用一维数组 p 来实现，数组下标 i 表示每个节点，p[i] 为指向其父节点的指针，即 p[x]=y 表示 x 的父节点为 y
- p 的初始值有两种设置方法，一种是均设为-1，另一种是令 p[i]=i, 两种方式在判断是否查找到树根有所不同, 设成-1 还有一个用处，可以用它来统计集合成员的数量

无向图的连通分量



边	构成的不相交集合									
初始	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b,d)	{a}	{b,d}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(e,g)	{a}	{b,d}	{c}		{e,g}	{f}		{h}	{i}	{j}
(a,c)	{a,c}	{b,d}			{e,g}	{f}		{h}	{i}	{j}
(h,i)	{a,c}	{b,d}			{e,g}	{f}		{h,i}		{j}
(a,b)	{a,b,c,d}				{e,g}	{f}		{h,i}		{j}
(e,f)	{a,b,c,d}				{e,f,g}			{h,i}		{j}
(b,c)	{a,b,c,d}				{e,f,g}			{h,i}		{j}

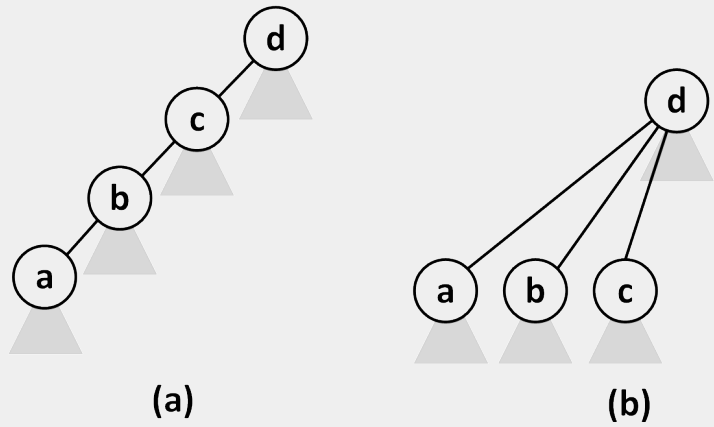
按秩合并(union by rank)

- 增加一个 Rank 数组（初始值为 0）来记录树的深度，也就是秩
- 将秩较小的树的根指向秩较大的树的根
- 任意顺序的合并操作以后，包含 k 个节点的树的最大高度不超过 $\log k$

```
1   void Union(int x,int  y)
2   {    x = Find(x);
3        y = Find(y);
4        if(x==y) return;
5        if(Rank[x] < Rank[y])
6             p[x]=y;
7        else {
8             p[y]=x;
9             if(Rank[x]==Rank[y])
10                 Rank[x]++;
11       }
12  }
```

- 路径压缩(path compression)
  - 每次查找的时候，可以将查找路径上的节点修改成指向根节点，以便下次查找的时候速度更快
  - 路径压缩的效果如图所示，(a) 为压缩前，(b) 为压缩后

```
1  int Find(int x)
2  {
3      if(p[x] < 0)
4          return x;
5      p[x] = Find(p[x]);
6      return p[x];
7  }
```



2524 – Ubiquitous Religions (poj.org)

- n 个学生信仰不同的宗教
- 给出 m 条信息，每条信息包含两个学生的编号，这两个学生信仰同一宗教
- 要求判断共有多少种宗教

6.2 线段树

- 线段树(Segment tree) 使用二叉树的结构，每个节点表示一个包含起点和终点的区间，也可以看成是一个线段
- 线段树对区间信息进行存储，可以实现一些与区间计算有关的操作，例如区间最值问题、区间求和问题等，计算几何里面扫描线的操作也可以用线段树实现
- 由于线段树消耗大量存储空间，熟练掌握离散化处理方法十分重要

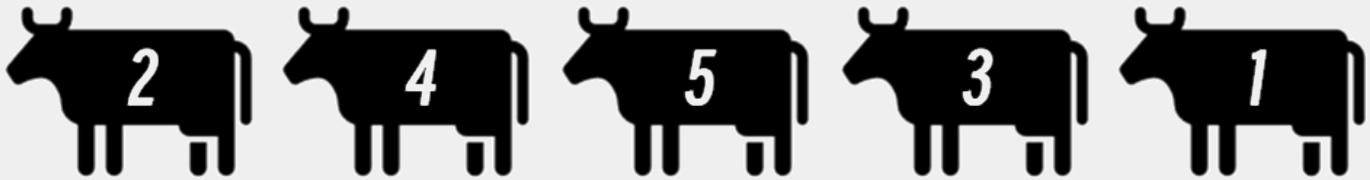
“在线”与“离线”

在程序设计过程中，如果开始时不需要知道所有输入，而是以序列的方式依次处理问题的算法，随着查询操作，数据也在实时变化，也称为“在线”查询，相应的算法称为在线算法，例如插入排序算法、贪心算法等

相对的，开始时就需要知道问题的所有数据，每次查询操作时的数据保持不变，称为“离线”查询，相应的算法称为离线算法。基于线段树的区间查询算法是离线查询，在多次查询的问题中能够提高效率

2182 – Lost Cows (poj.org)

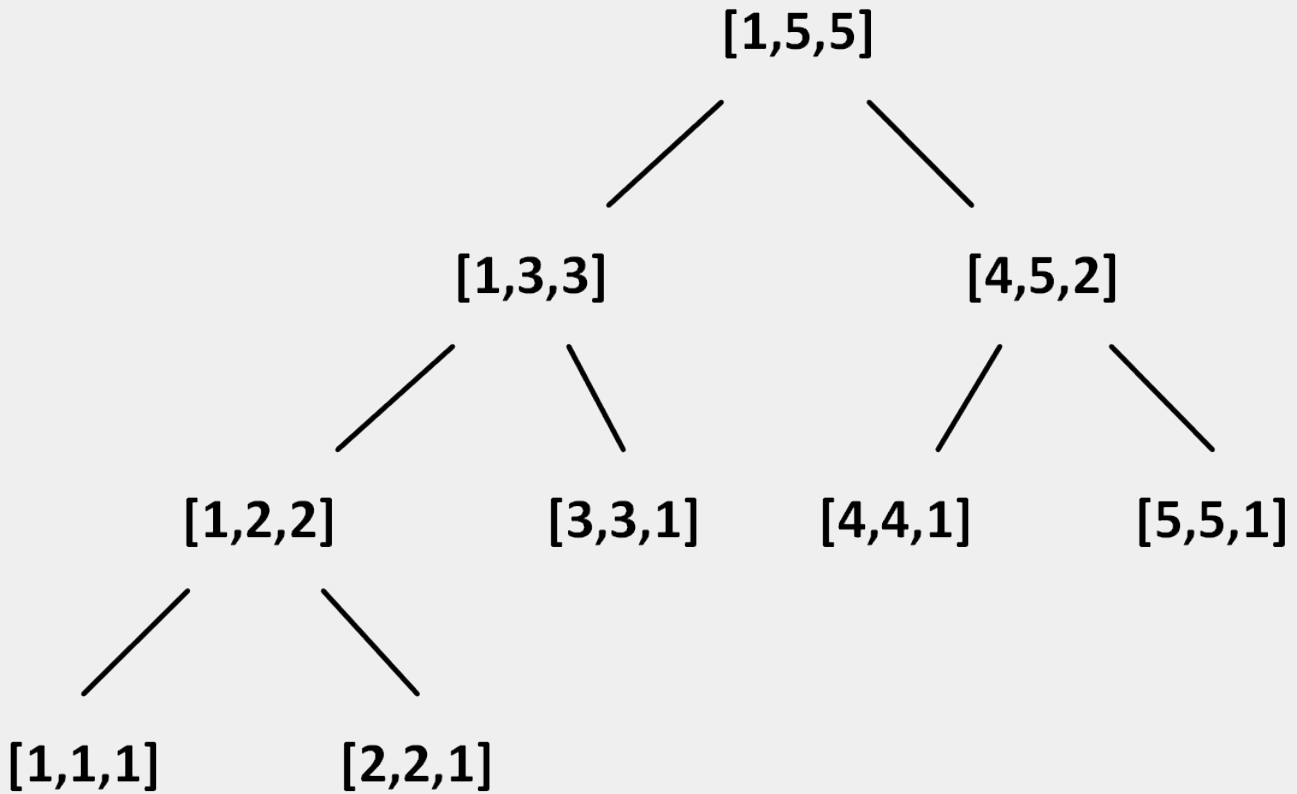
- n 头牛编号为 1...n，打乱顺序排成一列，除去队首的牛之外，给出每头牛在队列中前面编号比它小的牛的数量 k，求队列中每头牛的编号



- 维护一个线段树来解决该问题，创建一个线段树 T，树中每个节点有 3 个属性 $[p, r, n]$，分别代表线段左端点、线段右端点、左右端点之间节点的数量

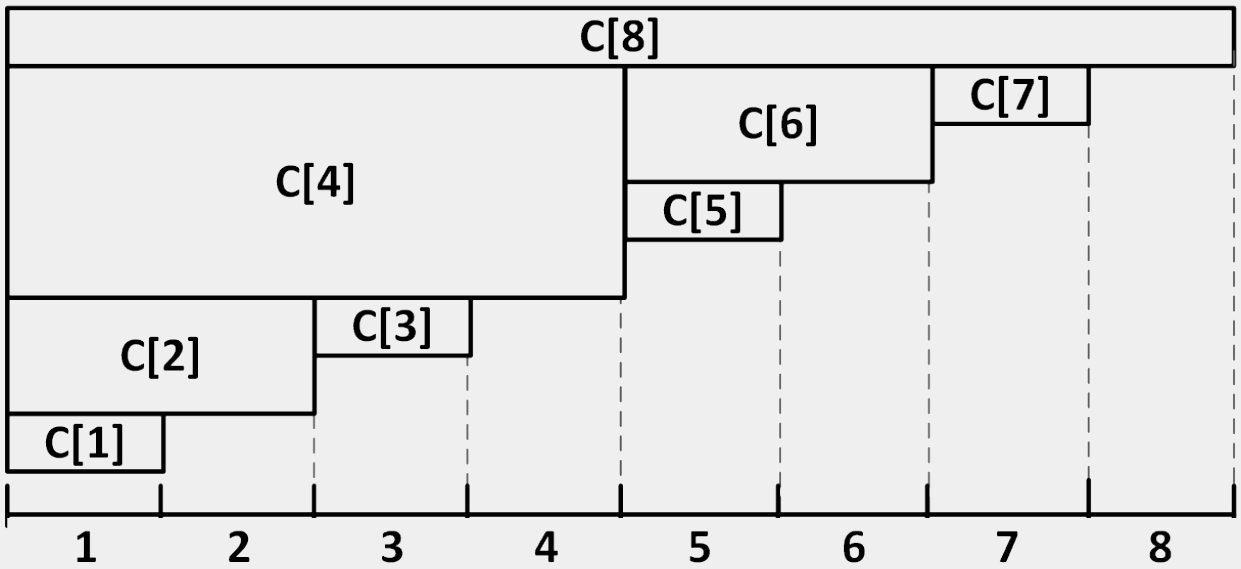
构造线段树 (n = 5)

- 从树根处依次比较：如果 k 小于等于左子树的 n，则在左子树中继续查询；如果 k 大于左子树的 n，则 $k = k - n$，在右子树中继续查询；直到找到叶子节点 ($p = r$)，此时的 p 即为要查询的编号



6.3 树状数组

- 树状数组也称二元索引树(Binary Indexed Tree) 或Fenwick 树，树状数组非常适合区间累计的计数与求和，尤其是多组查询，代码效率很高
- 与线段树相比，线段树可实现的功能更多，凡是树状数组能解决的问题，线段树同样可以解决
- 树状数组用一维数组 C 实现，每个元素代表不同区间（图中矩形横向范围）
  - C[1]:[1-1],C[2]:[1-2],C[3]:[3-3],C[4]:[1-4],C[5]:[5-5],C[6]:[5-6],...



树状数组

- 区间用下面方法确定：
  - 将下标用二进制表示出来，然后看末位有几个 0，设 0 的个数为 k，则它代表的区间就向前推 $2^k - 1$
  - 例如：$6 = (110)_2$，$k = 1$，向前推 $2^1 - 1 = 1$，所以表示的区间为 [5-6]
- 树状数组两个基本操作：

(1) 更新/添加元素 x：将 x 对应“列”的 C 值更新

- 例如：$x = 1$，需要将 $C[1], C[2], C[4], C[8]$... 都更新
- 计数：对应位置加 1
- 求和：对应位置加 x

(2) 区间求和：将对应区间“行”的 C 值相加

- 对于 $[1 \dots n]$ 区间求和，只需将对应“行”的 C 值相加，例如

$$\sum[1 \dots 7] = C[4] + C[6] + C[7]$$

- 对于 $[m \dots n]$ 区间求和，只需 $\sum[1 \dots n]$ 减去 $\sum[1 \dots m - 1]$，例如

$$\sum[3 \dots 5] = \sum[1 \dots 5] - \sum[1 \dots 2] = C[4] + C[5] - C[2]$$

lowbit

- 计算公式：*lowbit*(*x*) = *x*&(−*x*)，例如：*lowbit*(6)

十进制	二进制 (32 位 int 类型)
6	00000000000000000000000000000110
-6	11111111111111111111111111111010
&	00000000000000000000000000000010

- 计算出来的 *lowbit* 值见下表

x	1	2	3	4	5	6	7	8
lowbit(x)	1	2	1	4	1	2	1	8

- 观察树状数组的两个基本操作可以发现，下标的变化值为上一个下标的 *lowbit* 值
  (1) “更新”的列的下标变化，例如 1 对应列 C 的下标为 1,2,4,8,…，变化值为 *lowbit*(1), *lowbit*(2), *lowbit*(4), ...
  (2) “求和”的行的下标变化，例如 [1-7] 对应行 C 的下标为 7,6,4，变化值为 *lowbit*(7), *lowbit*(6)

2352 – Stars (poj.org)

- 给出 n 个星星的坐标 (x, y)，按 y 递增，y 相等 x 递增的次序
- 一颗星星的 level 定义为所有 x 和 y 均不大于该星星坐标的星星的总数，如图所示，编号 1…5 的星星的 level 分别为 0,1,2,1,3
- 依次输出 level 为 0…n-1 的星星个数，level 为 0 的有 1 个，level 为 1 的有 2 个，依次类推，所以输出 1,2,1,1,0

- 本题中给出的星星坐标已经将 y 递增排序，统计每个星星 x 坐标之前有多少个星星即可，也就是每输入一个星星的坐标，level 值就是 [1…x-1] 的累计值