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

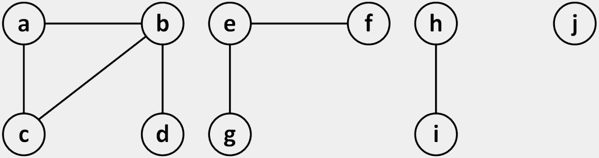
6. 高级数据结构

- 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



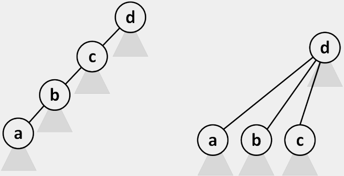
边	构成的不相交集合									
初始	{a}	{b}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(b,d)	{a}	{b,d}	{c}	{d}	{e}	{f}	{g}	{h}	{i}	{j}
(e,g)	{a}	{b,d}	{c}		{e,g}	{f}		{h}	{i}	{j}
(a,c)	{a,c}	{b,d}			{e,g}	{f}		{h}	{i}	{j}
(h,i)	{a,c}	{b,d}			{e,g}	{f}		{h,i}		{j}
(a,b)	{a,b,c,d}				{e,g}	{f}		{h,i}		{j}
(e,f)	{a,b,c,d}				{e,f,g}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

```
void Union(int x,int  y)
{     x = Find(x);
      y = Find(y);
      if(x==y) return;
      if(Rank[x] < Rank[y])
            p[x]=y;
      else {
            p[y]=x;
            if(Rank[x]==Rank[y])
                  Rank[x]++;
      }
}
```

- 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

```
int Find(int x)
{
    if (p[x] < 0)
        return x;
    p[x] = Find(p[x]);
    return p[x];
}
```



(a) (b)

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

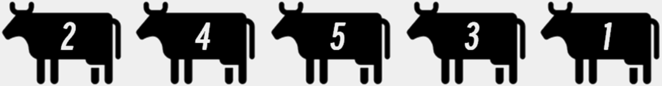
- 线段树(Segment tree) 使用二叉树的结构，每个节点表示一个包含起点和终点的区间，也可以看成是一个线段
- 线段树对区间信息进行存储，可以实现一些与区间计算有关的操作，例如区间最值问题、区间求和问题等，计算几何里面扫描线的操作也可以用线段树实现
- 由于线段树消耗大量存储空间，熟练掌握离散化处理方法十分重要

“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

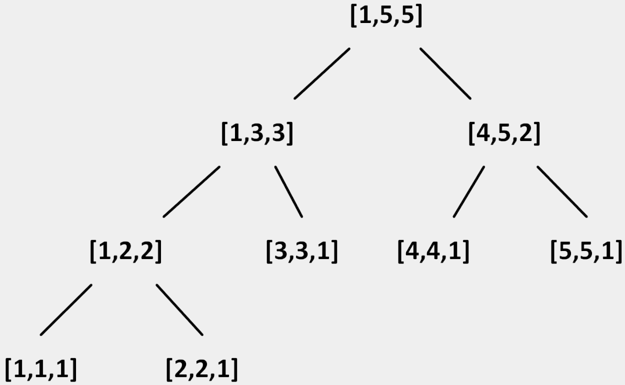
- n 头牛编号为 $1 \ldots n$，打乱顺序排成一列，除去队首的牛之外，给出每头牛在队列中前面编号比它小的牛的数量 k，求队列中每头牛的编号



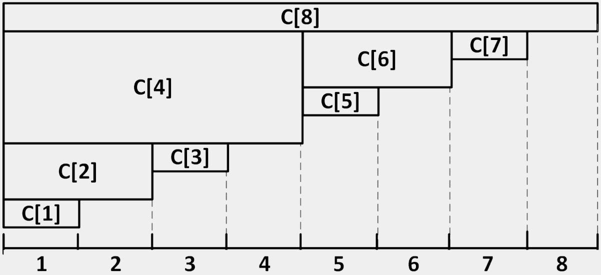
- 维护一个线段树来解决该问题，创建一个线段树 T，树中每个节点有 3 个属性 $[p, r, n]$，分别代表线段左端点、线段右端点、左右端点之间节点的数量

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



- 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 $lowbit(1), lowbit(2), lowbit(4), \ldots$
  (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

- 二叉搜索树(BST, Binary search tree) 具有如下性质：
  (1) root 是二叉搜索树的节点，x 是 root 节点的值
  (2) 如果 left 是 root 左子树的节点，则 left.x≤root.x
  (3) 如果 right 是 root 右子树的节点，则 right.x≥root.x

- 一般的二叉搜索树在最坏情况下，将长度为 n 的有序序列插入后，将变成一个长度为 n 的链表，查找效率降低

- 为了解决这个问题，产生了各种平衡树，常见的有**AVL 树**，树堆(Treap)，伸展树(Splay tree)，红黑树(Red-black tree) 等

2418 – Hardwood Species (poj.org)

- 给出一个单词列表，将这些单词去重之后按字典序排序，并计算出每个单词出现的次数在总单词数中的比例

- 使用二叉搜索树来存储单词和查找，树中每个节点包含左节点、右节点、单词、单词出现的次数这几个属性

- C++ STL 中的关联容器 map[1]就是使用二叉搜索树实现，map 内部通过自建一棵红黑树，实现了数据的自动排序和查找

[1] https://cplusplus.com/reference/map/map/

树堆

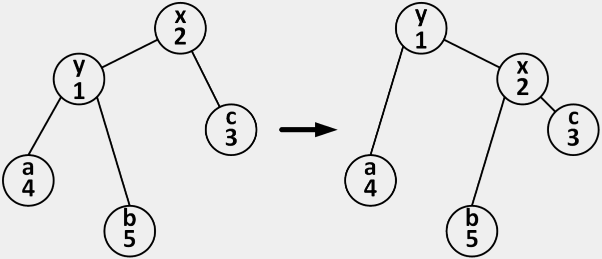
- 对于二叉搜索树，在插入过程中如果插入的值一直小于或大于上一个值，树会向一个方向伸展，这会造成二叉搜索树不平衡，而降低搜索效率，这时可以使用树堆来解决该问题

树堆 (Treap)

通过添加一个随机值 r 来解决二叉搜索树不平衡问题，树堆根据 r 值，保持树的节点的层次位置按 r 值单调排列，因为 r 是随机的，树的节点就会保证基本的平衡，r 值各不相等的情况下，生成的树是唯一的

树堆

- 树堆的插入、查找和删除等基本操作和普通的二叉搜索树类似，这里关键的操作就是根据 r 进行树的调整
  - 树的每个节点上方为标识，下方为 r 值 (从上到下依次递增)，从左到右节点 a, b, x, c 的 p 值依次递增
  - 插入一个新节点 y，它的 p 值小于根节点 x，插入 x 左侧，插入以后如左图，整个树的 r 值不符合要求，需要进行旋转操作
  - 树堆的调整分为右旋操作和左旋操作，左节点 r 值小于根节点 r 值，执行右旋操作，执行完如右图，树的形态发生了变化，但是 p 值的顺序不变



3481 – Double Queue (poj.org)

- 系统为客户服务，每个客户有一个标识 k，一个优先级 p。这个系统可以接收的请求和对应的策略如下：

请求	策略
0	停止服务
1 k p	添加优先级 p 的客户 k
2	服务优先级最高的客户并移除等待队列
3	服务优先级最低的客户并移除等待队列

散列表

散列表 (Hash table，也叫哈希表) 基本原理是使用一个下标范围比较大的数组 $H[1 \ldots m]$ 来存储元素 x

- 设计一个哈希函数，使得每个 x 都与一个函数值 y(即数组 H 的下标) 相对应，然后用 $H[y]$ 指向元素 x，没有元素指向，则 $H[y] = \phi$
- 由于不能够保证每个元素的关键字与函数值是一一对应的，因此很有可能出现如下情况：对于不同的元素 x_1, x_2，哈希函数计算出了相同的函数值 y，这就是产生了所谓的 "冲突"，换句话说，就是哈希函数同一个 $H[y]$ 指向了不同的元素 x_1, x_2
- 常用的字符串哈希函数有 BKDRHash、PJWHash(ELFHash)、APHash 等，效率较高的为 BKDRHash 函数[2]

[2] https://byvoid.com/zhs/blog/string-hash-compare/

解决冲突

1. 链接表，将相同函数值的所有元素放在一个链表中，$H[y]$ 指向这个链表
2. 开放寻址法
   - 最简单的是线性探测再散列技术，即当 $H[y] \neq \phi$，说明 $H[y]$ 位置已经存储有元素，依次探测 $(y+i) \mod m, i = 1, 2, 3 \ldots$，直到找到空的存储单元为止
   - 当冲突严重时，扫描到空单元的时间会变长，哈希表越满，这种情况带来时间消耗就会越大，这时通过扩大数组范围 m 可以有效减少查找时间
   - 设计一个好的哈希函数是解题的关键，而实际应用中，哈希函数的种类很多，可以应用到多个领域，理解哈希函数的作用十分必要

2503 – Babelfish (poj.org)

- 给出字典条目分为 < 英语外语 >，接下来给出外语，在字典中查找对应的英语并输出，如果查不到，输出“eh”
- 采用哈希表解决，将字符串转换成对应的哈希值，这样在查找时会提高效率
- 将外语词条保存到一个哈希表 H 中，英语保存到对应的表 A 中

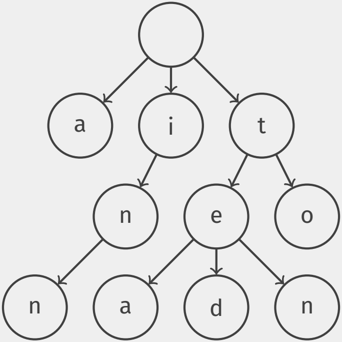
  (1) 向哈希表添加元素：通过一个字符串哈希函数将外语字符串 x 转化成整数 y，将 x 保存到 $H[y]$ 中，将 x 对应的英语保存到 $A[y]$ 中。如果冲突，按照线性探测再散列的方法，令 $y = y + i(i = 1, 2, ...)$

  (2) 在哈希表中查找元素：当需要查找外语字符串 x，先通过哈希函数计算出 x 的哈希值 y，读出 $H[y]$ 的值，将 $H[y]$ 和 x 比较，如果相等，则对应的 $A[y]$ 就是要求的内容；如果不相等，说明之前 $H[y]$ 发生冲突，存储了其他的字符串的位置，这时需要在 $y + i(i = 1, 2, ...)$ 位置继续查询;$H[y] = \phi$ 表明没有找到
- 越接近单词总量，哈希表剩余的空位置越少，由于查找是到下一个空位置终止，终止前花费的查找时间就会越长

- 字符串一个常见的应用就是模式匹配问题，比如文本编辑软件中的查找功能，网页搜索，GNU 的 grep 命令等
- 将字符串扩展到大量的文本，需要更高效的数据结构和算法来实现
- 除了Trie 树、KMP 算法之外，还有后缀树(Suffix tree)，后缀数组(Suffix array)，Boyer-Moore 字符串搜索算法，在文本里搜索多个字符串的Aho-Corasick 算法(AC 自动机) 等

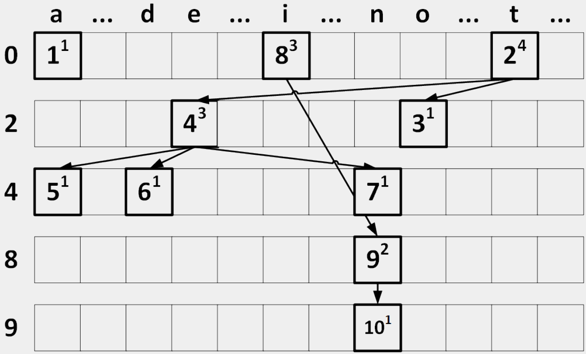
字典树

- Trie 树，也称前缀树或字典树
  - 例如："a", "to", "tea", "ted", "ten", "i", "in", "inn" 构成的 trie 树如图所示。根节点为空字符串，每个字符串从根节点到对应的叶子节点，每个节点的子孙都有相同的前缀
  - trie 树使用链表存储，也可以使用二维数组，二维数组在空间上有所浪费，但是建树和查找过程效率很高



字典树

- 使用二维数组建树后的内容如图所示。数组 trie[i,j] 中存储的值为字符插入树的顺序 pos，从根节点到 pos 的路径就是 pos 后面包含字符串的公共前缀；图中数字上标为数组 num[pos]，其值为多少字符串以根到 pos 为前缀



2001 – Shortest Prefixes (poj.org)

- A prefix of a string is a substring starting at the beginning of the given string. The prefixes of ”carbon” are: ”c”, ”ca”, ”car”, ”carb”, ”carbo”, and ”carbon”. Note that the empty string is not considered a prefix in this problem, but every non-empty string is considered to be a prefix of itself.
- In everyday language, we tend to abbreviate words by prefixes. For example, ”carbohydrate” is commonly abbreviated by ”carb”.
- In this problem, given a set of words, you will find for each word the shortest prefix that uniquely identifies the word it represents.

3461 – Oulipo (poj.org)

- 题意：给出字符串 W 和 T，求 W 在 T 中出现的次数
- Knuth-Morris-Pratt 字符串查找算法 (KMP 算法)：基于前缀匹配的方法，可以提高匹配效率
- 如图所示，当 P[6]≠S[6] 时，可以退回到 P[2] 继续匹配，用一个数组 next 记录回退的位置，比如 next[6]=2，就可以通过该数组实现回退
- next 数组的求法相当于在 P 数组内部进行字符串的匹配，值就是匹配到的共同前缀长度

```
S:  A B C A B C A B D
P:  A B C A B D
S:  A B C A B C A B D
P:        A B C A B D
```