Data Structure I.

Basic Manual Data Structure by Example

trswnca @ tjuacm

2021 TJU Summer Camp

September 23, 2021

Preface

数据结构是在计算机中存储、组织数据的方式。
此处数据结构更加偏指通过一些维护操作，使得数据具有更利于答案计算的性质。

鉴于过于简单，不会的自学

- Queue: `https://oi-wiki.org/ds/stack/`
- Stack: `https://oi-wiki.org/ds/queue/`
- Linked list `https://oi-wiki.org/ds/linked-list/`

具体的应用可以看《算法导论》

Data Structure

$fa[i]$ 表示 i 的父亲
$1 \sim n$ 构成一个森林，每颗树代表一个集合

Initialization

```
const int N = 1e5 + 10;
int fa[N];
void init() { fa[i] = i; }
```

Basic Operations

Find

```
int find(int x) {
  return x == fa[x] ? x : find(fa[x]);
}
```

Union

```
void union(int x, int y) {
  x = find(x), y = find(y);
  fa[x] = y;
}
```

Tech

并查集可以维护具有**传递性**的数据
如果按照不做优化，单次查找会遍历整条树链，复杂度 $O(n)$
一般有两种优化：

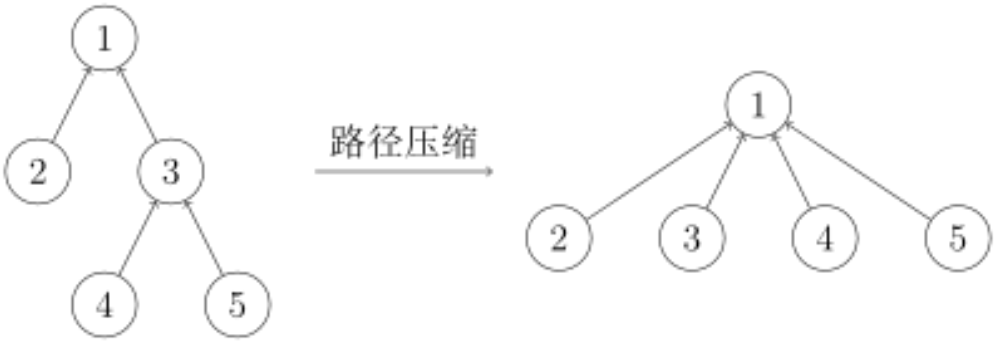
- 路径压缩
- 按秩合并

更多优化：
https://www.zhihu.com/question/28410263/answer/40966441

Compression

find

```
int find(int x) {
  return x == fa[x] ?
    x : fa[x] = find(x);
}
```



这样复杂度为 $O(\alpha(m,n)n)$ 其中 $\alpha(m,n)$ 为反阿克曼函数，一般可以认为小于 5

Linking by rank

我们将一棵点数与深度都较小的集合树连接到一棵更大的集合树下

union

```
void union(int x, int y) {
  x = find(x), y = find(y);
  if (x == y) return;
  if (size[x] > size[y]) swap(x, y);
  fa[x] = y; size[y] += size[x];
}
```

BZOJ 3376

Cube Stacking

约翰和贝茜在玩一个方块游戏。编号为 1 到 n 的 n 个方块正放在地上. 每个构成一个立方柱。游戏开始后，约翰会给贝茜发出 m 个指令。指令有两种：

- 移动：将包含 x 的立方柱移动到包含 y 的立方柱上；
- 统计：统计名含 x 的立方柱中，在 x 下方的方块数目。

BZOJ 3376

Sol.

带权并查集，以每堆方块最下面的方块为并查集的树根，记录每堆方块最上面的方块编号，每次合并将 x 块的根的父节点置为 y 所在堆最上面的方块。
记 $d(x)$ 为从 x（包含）到 x 的父节点（不包含）之间的方块数量。每次查询将 $d(x)$ 取一个前缀和即可。

Implement: `https://vjudge.net/solution/31828472`

UVA 11987

Almost Union-Find

n 个数，从 1 到 n，初始状态分属不同集合

- 合并数字 p 和数字 q 所在的集合
- 把 p 插入 q 所在的集合（并在 p 原来所在的集合中删除 p）
- 询问某个数字 p 所在集合的元素个数以及总和

UVA 11987

Sol .

做一次 镜像 $f[i] = i + n, f[i + n] = i + n$

Introduction

左偏树是一种 **可并堆**，具有堆的性质，并且是「左偏」的，因此可以快速合并。

Introduction

左偏树是一种 **可并堆**，具有堆的性质，并且是「左偏」的，因此可以快速合并。

对于一棵二叉树，我们定义 **外节点**为左儿子或右儿子为空的节点，定义一个外节点的 *dist* 为 1，一个不是外节点的节点 *dist* 为其到子树中最近的外节点的距离加一。**空节点的** *dist* **为** 0 。

「左偏」：每个节点左儿子的 *dist* 都大于等于右儿子的 *dist* 。

Data Structure

Data

```
struct Node {
  int ls, rs, val, d;
}
```

Merge

Merge

```
int merge(int x, int y) {
  if (!x || !y) return x | y;
  if (t[x].val > t[y].val) swap(x, y);
  t[x].rs = merge(t[x].rs, y);
  if (t[t[x].rs].d > t[t[x].ls].d)
    swap(t[x].ls, t[x].rs);
  t[x].d = t[t[x].rs].d + 1;
  return x;
}
```

Merge

由于左偏性质，每递归一层，其中一个堆根节点的 dist 就会减小 1，而“一棵有 n 个节点的二叉树，根的 dist 不超过 $\lceil \log(n+1) \rceil$”，所以合并两个大小分别为 n 和 m 的堆复杂度是 $O(\log n + \log m)$。

Extended

- 删除根节点

Merge

由于左偏性质，每递归一层，其中一个堆根节点的 dist 就会减小 1，而“一棵有 n 个节点的二叉树，根的 dist 不超过 $\lceil \log(n+1) \rceil$”，所以合并两个大小分别为 n 和 m 的堆复杂度是 $O(\log n + \log m)$。

Extended

- 删除根节点 合并根的左右儿子
- 插入节点

Merge

由于左偏性质，每递归一层，其中一个堆根节点的 dist 就会减小 1，而“一棵有 n 个节点的二叉树，根的 dist 不超过 $\lceil \log(n+1) \rceil$”，所以合并两个大小分别为 n 和 m 的堆复杂度是 $O(\log n + \log m)$。

Extended

- 删除根节点 合并根的左右儿子
- 插入节点 相当于合并只有一个节点的堆

Modify

可以维护不改变相对大小的对整个堆的修改，在根打上标记，删除根/合并堆（访问儿子）时下传标记即可

Modify

```
int merge(int x, int y) {
  if (!x || !y) return x | y;
  if (t[x].val > t[y].val) swap(x, y);
  pushdown(x);
  t[x].rs = merge(t[x].rs, y);
  if (t[t[x].rs].d > t[t[x].ls].d) swap(t[x].ls, t[x].rs);
  t[x].d = t[t[x].rs].d + 1;
  return x;
}
int pop(int x) {
  pushdown(x); return merge(t[x].ls, t[x].rs);
}
```

Luogu P1456

Monkey King

曾经在一个森林中居住着 N 只好斗的猴子。在最初他们互不认识。争吵发生时，双方会邀请它们各自最强壮的朋友并发起决斗。在决斗之后两只猴子和他们各自的伙伴都成为朋友. 假设每只猴子有一个强壮值，强壮值将在一场决斗后减少为原先的一半. 给出 M 次争斗，询问每次争斗后最强壮的值。

Luogu P1456

Monkey King

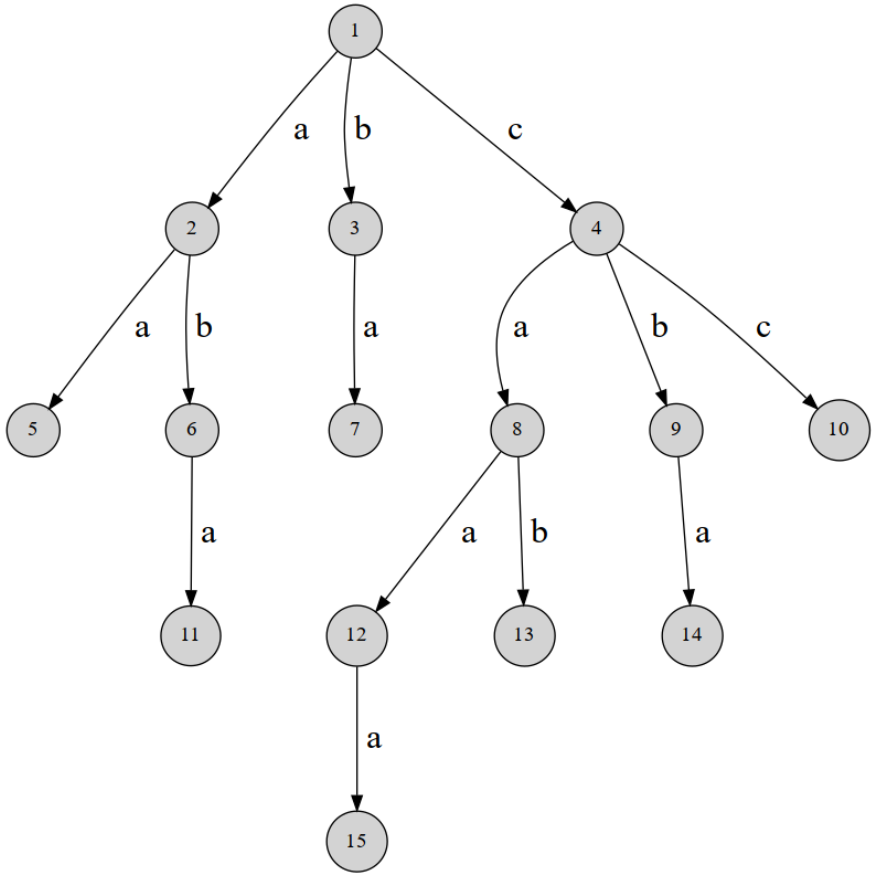
曾经在一个森林中居住着 N 只好斗的猴子。在最初他们互不认识。争吵发生时，双方会邀请它们各自最强壮的朋友并发起决斗。在决斗之后两只猴子和他们各自的伙伴都成为朋友. 假设每只猴子有一个强壮值，强壮值将在一场决斗后减少为原先的一半. 给出 M 次争斗，询问每次争斗后最强壮的值。

Sol .

可合并的大根堆，删除根节点，插入根节点的一半

Implement: `https://www.luogu.com.cn/record/54354821`

Trie 树是用链代表字符串，节点维护信息的树



Trie

Data Structure

Data

```
const int N = 1e5 + 10;
const int M = 26;
int ch[N * M][M], tot;
```

$ch[x][i]$ 存 x 节点的按字典序第 i 个儿子
如果子节点为空，则 $ch[x][i] = 0$

Operations

Insert

```
void insert(char s[]) {
  int n = strlen(s), x = 0;
  for (int i = 0; i < n; ++ i) {
    if (!ch[x][s[i] - 'a']) ch[x][s[i] - 'a'] = ++tot;
    x = ch[x][s[i] - 'a']; val[x] ++;
  }
}
```

val 可以维护子树内信息，具有前缀和性质的信息

- 前缀个数
- 子树最小值
- ...

Operations

Query

```
bool query(char s[]) {
  int n = strlen(s), x = 0;
  for (int i = 0; i < n; ++ i) {
    if (!ch[x][s[i] - 'a']) return false;
    x = ch[x][s[i] - 'a'];
  }
  return true;
}
```

询问的值可以根据 *val* 的设置调整

Luogo P4551

最长异或路径

给定一棵 n 个点的带权树，结点下标从 1 开始到 N。寻找树中找两个结点，求最长的异或路径。
异或路径指的是指两个结点之间唯一路径上的所有边权的异或。

Luogo P4551

最长异或路径

给定一棵 n 个点的带权树，结点下标从 1 开始到 N。寻找树中找两个结点，求最长的异或路径。
异或路径指的是指两个结点之间唯一路径上的所有边权的异或。

Sol.

通过异或前缀和求一段连续区间异或
到根节点只有 n 个前缀，全部插入，再走一遍进行查询即可
Implement: `https://www.luogu.com.cn/record/53774710`

Introduction

树状数组就是用数组来模拟树形结构，因为写法简洁，所以很多能使用树状数组解决的问题不用建树解决.
树状数组可以解决大部分基于区间上的更新以及求和问题，修改和查询的复杂度都是 $O(logN)$.

Data Structure

Data

```
const int N = 1e5 + 10;
int C[N];
```

$C[x]$ 储存

$$\sum_{k=x-\text{lowbit}(x)}^{x} a[k]$$

$\text{lowbit}(x) = x\& - x$ 表示二进制表示下，最低位为 1 的数

Query

Query

这里说的查询是查询任一区间的和，由于区间和具有可加减性，故转化为求前缀和；
查询前缀和就是把大区间分成几段长度不等的小区间，然后求和。区间的个数为 $O(logn)$，所以查询的时间复杂度为 $O(logn)$。

```
int query(int *C, int n, int x) {
  int res = 0;
  for (; x > 0; x -= x & -x) res += C[x];
  return res;
}
```

Add

Add

更新的时候只要更新修改某个点会影响到哪些数组

```
void update(int *C, int n, int x, int val) {
  for (; x <= n; x += x & -x) C[x] += val;
}
```

LOJ 130, 131, 132

树状数组模板 3 题

- 130：单点修改，单点询问
- 131：区间修改，单点查询
- 132：区间修改，区间查询

LOJ 130, 131, 132

树状数组模板 3 题

- 130：单点修改，单点询问
- 131：区间修改，单点查询
- 132：区间修改，区间查询

Sol.

- 130：**Implement:** `https://loj.ac/s/858312`
- 131：差分后求和即可
  Implement: `https://loj.ac/s/858367`
- 132：维护两个差分数组
  Implement: `https://loj.ac/s/858505`

Data Structure

线段树维护一个区间 (且能区间合并) 的信息，所以一般需要维护 **是哪个区间 区间信息**

线段树基础

如果空间要求比较严苛，可以省略区间 $[l, r]$ 的记录，递归时带区间范围

Data

```
const int N = 1e5 + 10;
struct Node {
  int l, r, val, tag;
} t[N << 2];
```

Problem A

问题

维护一个长度为 n 的序列：

- 区间 $[l, r]$ 加上 x
- 区间 $[l, r]$ 乘上 x
- 查询区间和

Problem A

问题

维护一个长度为 n 的序列：

- 区间 $[l, r]$ 加上 x
- 区间 $[l, r]$ 乘上 x
- 查询区间和

先加再乘还是先乘再加？

先加再乘

$(val, add, mul) \odot (ad, mu)$

$(val + add) \times mul \longrightarrow [(val + add) \times mul + ad] \times mu$

先加再乘

$(val, add, mul) \odot (ad, mu)$

$(val + add) \times mul \longrightarrow [(val + add) \times mul + ad] \times mu$ 那么

$$\begin{cases} add' &= add + ad/mu \\ mul' &= mul \times mu \end{cases}$$

先乘再加

$$(val, add, mul) \odot (ad, mu)$$
$$val \times mul + add \times (r - l + 1) \longrightarrow (val \times mul + add \times (r - l + 1)) \times mu + ad$$

先乘再加

$(val, add, mul) \odot (ad, mu)$

$val \times mul + add \times (r - l + 1) \longrightarrow (val \times mul + add \times (r - l + 1)) \times mu + ad$

那么

$$\begin{cases} add' &= add \times mu + ad \\ mul' &= mul \times mu \end{cases}$$

Problem B

维护一个长度为 n 的序列：

- 区间 $[l, r]$ 加上 a_i
- 区间 $[l, r]$ 加上 1
- 区间 $[l, r]$ 置为 x
- 查询区间和

Sol .

val 当前节点表示的区间和
*add*1 第一种操作值
*add*2 第二种操作值
flag 是否覆盖，不覆盖为 -1

Luogu P3605

Promotion Counting P

给出一颗带权树，询问有多少数对，满足孩子的值大于祖先.

Luogu P3605

Promotion Counting P

给出一颗带权树，询问有多少数对，满足孩子的值大于祖先.

Sol .

如果一颗维护了子树的权值信息，那么对于权值为 x 的点，只需要查询 $x+1 \sim n$ 的数量即可.
如果自底向上递归，不断合并线段树维护子树权值，即可求出所有数量.
具体来说，只需要选择一个节点，把另外一个节点的值加过来，即可完成
Implement:
`http://blog.trswnca.top/index.php/archives/72/`

Extended

事实上，只要是能够区间合并的信息都可以用线段树维护
如：和、积、最值、矩阵乘积、gcd、线性基、bitset、hash

Thanks!