数据结构

hzwer

Peking University

2016 年 8 月 16 日

散列表（Hash table，也叫哈希表），是根据关键码值 (Key value) 而直接进行访问的数据结构。

散列表（Hash table，也叫哈希表），是根据关键码值 (Key value) 而直接进行访问的数据结构。
也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。

散列表（Hash table，也叫哈希表），是根据关键码值 (Key value) 而直接进行访问的数据结构。
也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。
这个映射函数叫做散列函数，存放记录的数组叫做散列表。

元素查找

给出 n 个正整数，然后有 m 个询问，每次询问一个整数 x，询问该整数是否在 n 个正整数中出现过。

元素查找

给出 n 个正整数，然后有 m 个询问，每次询问一个整数 x，询问该整数是否在 n 个正整数中出现过。

$0 \leq x \leq 10^9, 0 \leq n, m \leq 100000$

单数字散列

1. 直接取模

单数字散列

1. 直接取模
2. 各种运算

单数字散列

1. 直接取模
2. 各种运算

单词查找

给出 n 个单词，然后有 m 个询问，每次询问一个不超过 50 个字符的单词，询问该单词是否在 n 个单词中出现过。

单词查找

给出 n 个单词，然后有 m 个询问，每次询问一个不超过 50 个字符的单词，询问该单词是否在 n 个单词中出现过。

$0 \leq n, m \leq 100000$

单数字散列

1. 直接取模
2. 各种运算

单数字散列

1. 直接取模
2. 各种运算

数列/字符串的散列

1. 排序编号

单数字散列

1. 直接取模
2. 各种运算

数列/字符串的散列

1. 排序编号
2. 边乘边模

单数字散列

1. 直接取模
2. 各种运算

数列/字符串的散列

1. 排序编号
2. 边乘边模
3. 自然溢出

单数字散列

1. 直接取模
2. 各种运算

数列/字符串的散列

1. 排序编号
2. 边乘边模
3. 自然溢出

单数字散列

1. 直接取模
2. 各种运算

数列/字符串的散列

1. 排序编号
2. 边乘边模
3. 自然溢出

棋盘散列

冲突/碰撞解决

1. 链地址法/拉链法

冲突/碰撞解决

1. 链地址法/拉链法
   用链表将同一 hash 值的若干元素串联

冲突/碰撞解决

1. 链地址法/拉链法
   用链表将同一 hash 值的若干元素串联
2. 开放寻址法

冲突/碰撞解决

1. 链地址法/拉链法
   用链表将同一 hash 值的若干元素串联
2. 开放寻址法
   若某元素插入的散列地址已有元素占用，则使用一定方法寻找下一个可用地址

冲突/碰撞解决

1. 链地址法/拉链法
   用链表将同一 hash 值的若干元素串联
2. 开放寻址法
   若某元素插入的散列地址已有元素占用，则使用一定方法寻找下一个可用地址
3. 多重哈希

冲突/碰撞解决

1. 链地址法/拉链法
   用链表将同一 hash 值的若干元素串联
2. 开放寻址法
   若某元素插入的散列地址已有元素占用，则使用一定方法寻找下一个可用地址
3. 多重哈希

在一些有 N 个元素的集合应用问题中，我们通常是在开始时让每个元素构成一个单元素的集合
然后按一定顺序将属于同一组的元素所在的集合合并，其间要反复查找一个元素在哪个集合中

在一些有 N 个元素的集合应用问题中，我们通常是在开始时让每个元素构成一个单元素的集合
然后按一定顺序将属于同一组的元素所在的集合合并，其间要反复查找一个元素在哪个集合中
这一类问题数据量极大，若用正常的数据结构来描述的话，计算机无法承受

在一些有 N 个元素的集合应用问题中，我们通常是在开始时让每个元素构成一个单元素的集合
然后按一定顺序将属于同一组的元素所在的集合合并，其间要反复查找一个元素在哪个集合中
这一类问题数据量极大，若用正常的数据结构来描述的话，计算机无法承受
只能采用一种全新的抽象的特殊数据结构——并查集来描述

- 初始化
  把每个点所在集合初始化为其自身
  通常来说，这个步骤在每次使用该数据结构时只需要执行一次，无论何种实现方式，时间复杂度均为 $O(n)$

- 初始化
  把每个点所在集合初始化为其自身
  通常来说，这个步骤在每次使用该数据结构时只需要执行一次，无论何种实现方式，时间复杂度均为 $O(n)$
- 查找
  查找元素所在的集合，即根节点

- 初始化
  把每个点所在集合初始化为其自身
  通常来说，这个步骤在每次使用该数据结构时只需要执行一次，无论何种实现方式，时间复杂度均为 $O(n)$
- 查找
  查找元素所在的集合，即根节点
- 合并
  将两个元素所在的集合合并为一个集合
  通常来说，合并之前，应先判断两个元素是否属于同一集合，这可用上面的“查找”操作实现

顾名思义，就是在维护元素所属集合的同时记录集合的一些信息

- 维护集合的最大最小值
- 维护集合的和
- 维护其它的信息

- 路径压缩
- 按秩合并
- 启发式合并

冗余关系

n 个人，互相 m 个朋友关系
且满足：朋友的朋友是我的朋友
求冗余关系的数目
冗余关系定义为：即使没有这个关系，原图的所有关系都成立

冗余关系

n 个人，互相 m 个朋友关系
且满足：朋友的朋友是我的朋友
求冗余关系的数目
冗余关系定义为：即使没有这个关系，原图的所有关系都成立
$0 \leq n, m \leq 10^6$

题解

依次连边，如果有一条边的两个端点在同一个连通块，则该关系冗余

mty 的考验

n 个人，互相 m 个朋友关系，要组建一个军队
必须满足军队中的每个人之间都有直接或间接的朋友关系，组建一支在满足上述情况下的人数最多的军队

mty 的考验

n 个人，互相 m 个朋友关系，要组建一个军队
必须满足军队中的每个人之间都有直接或间接的朋友关系，组建一支在满足上述情况下的人数最多的军队
$0 \leq n, m \leq 10^6$

题解

依次连边，带权并查集维护连通块的大小

团伙

n 个强盗。如果两个强盗遇上了，那么他们要么是朋友，要么是敌人。而且：

1. 我的朋友的朋友是我的朋友。
2. 我敌人的敌人也是我的朋友。

两个强盗是同一团伙的条件是当且仅当他们是朋友。

现在给定 m 条强盗之间的关系，问最多有多少个强盗团伙。

保证输入的强盗关系的合法性

团伙

n 个强盗。如果两个强盗遇上了，那么他们要么是朋友，要么是敌人。而且：

1. 我的朋友的朋友是我的朋友。
2. 我敌人的敌人也是我的朋友。

两个强盗是同一团伙的条件是当且仅当他们是朋友。

现在给定 m 条强盗之间的关系，问最多有多少个强盗团伙。

保证输入的强盗关系的合法性

$0 \leq n \leq 100000, 0 \leq m \leq 1000000$

题解

将有关系的强盗并集

题解

将有关系的强盗并集
若 2 个强盗是朋友，则连一条边权为 0 的边

题解

将有关系的强盗并集
若 2 个强盗是朋友，则连一条边权为 0 的边
若 2 个强盗是敌人，则连一条边权为 1 的边

题解

将有关系的强盗并集
若 2 个强盗是朋友，则连一条边权为 0 的边
若 2 个强盗是敌人，则连一条边权为 1 的边
若 2 个强盗“有关系”，则这 2 个强盗属于同一个连通图，反之亦然

题解

将有关系的强盗并集
若 2 个强盗是朋友，则连一条边权为 0 的边
若 2 个强盗是敌人，则连一条边权为 1 的边
若 2 个强盗“有关系”，则这 2 个强盗属于同一个连通图，反之亦然
2 个强盗之间的任意一条路径阐述了他们之间的关系

题解

将有关系的强盗并集
若 2 个强盗是朋友，则连一条边权为 0 的边
若 2 个强盗是敌人，则连一条边权为 1 的边
若 2 个强盗“有关系”，则这 2 个强盗属于同一个连通图，反之亦然
2 个强盗之间的任意一条路径阐述了他们之间的关系
将路径上的所有边权加起来，对 2 取模，为 0 则是朋友，为 1 则是敌人

题解

将有关系的强盗并集
若 2 个强盗是朋友，则连一条边权为 0 的边
若 2 个强盗是敌人，则连一条边权为 1 的边
若 2 个强盗“有关系”，则这 2 个强盗属于同一个连通图，反之亦然
2 个强盗之间的任意一条路径阐述了他们之间的关系
将路径上的所有边权加起来，对 2 取模，为 0 则是朋友，为 1 则是敌人

环

给出一张 n 个点 m 条边的无向图，依次询问某个点所在连通块是否为环，是否为树?
(链也算树)

环

给出一张 n 个点 m 条边的无向图，依次询问某个点所在连通块是否为环，是否为树?
(链也算树)
$0 \leq n, m \leq 10^6$

题解

依次连边，当一条边连接的两个点处于同一连通块时
根据出入度判断连通块的形状

链型网络

给定一张无重边，自环的无向图.
每次可以加边，或者询问有多少个点满足将该点删除后，原图的每个连通块都为一条链

直观感受

对于下面一些简单的情况，我们可以容易得出答案.

- 原图为若干条链，则答案为点数 N;
- 原图为单个简单环加若干条链，则答案为环大小;
- 原图中超过一个连通块有环，答案为 0.
- 原图中一个连通块为一个点连接三条链，答案为 4;
- 原图中一个连通块为一个点连接三条以上的链，答案为 1;

但是考虑到环套树这样更复杂的情况，上述分类讨论也无能为力.

但是考虑到环套树这样更复杂的情况，上述分类讨论也无能为力.

我们需要思考链本身的性质，一个图的每个连通块为链，等价于每个点的度数小于等于 2 且无环. 转化后的条件明显更有利于解决问题.

首先考虑图中是否有度数大于等于 3 的点，如果存在一个度数大于等于 3 的点 u，因为要保证去掉一个点后每个点的度数都小于等于 2，我们要么去掉 u，要么在 u 度数恰好为 3 时，去掉 u 的三个相邻点中的一个. 而其他的点均不可能成为答案.

这样我们只需要在加边过程中第一次出现度数为 3 的点的时候，对于可能成为答案的 4 个点，分别维护一个去掉该点之后的图.
然后在 4 个图中分别进行判断
只要在加边时判断每个点度数都小于等于 2, 再用并查集判断是否有环即可.

剩下的就是每个点的度数都小于等于 2 的情况，这样每个连通块只能是链或者简单环，我们只需采用一开始的分类讨论即可.

- 原图为若干条链，则答案为点数 N;
- 原图为单个简单环加若干条链，则答案为环大小;
- 原图中超过一个连通块有环，答案为 0.

这只需要维护一个记录集合大小的并查集就能做到.