**排序**

1. **实验目的和要求**

由于按高度并和路径压缩很难同时使用。因为路径压缩改变了树的高度，而现在又很难计算新的高度。为此，提出了按秩并的概念。用秩作为高度的估计值。在路径压缩后，并不改变秩的大小。试实现采用按秩并和路径压缩的不相交集类。

**二、实验内容**

1. 实验准备
   1. 理论知识介绍
      1. 等价关系与等价类
         1. 等价关系的定义

对集合中的每一对元素(a,b), a,b∈S, aRb要么是真要么是假，则称关系R（relation R）是定义在集合S上。如果aRb为真，则称a与b相关。如果集合中的每一对元素不是有关系就是没关系，则称该关系（relation）是定义在该集合上的。

* + - 1. 等价关系的特性

等价关系（equivalence relation）是一种满足以下三个特性关系R。

自反性（Reflexive）：对所有的a∈R，aRa为真。

对称性（Symmetric）：当且仅当bRa时，aRb。

传递性（Transitive）：aRb和bRc隐含aRc。

* + - 1. 等价关系的储存

二维数组：

这是最直接的方法。i和j有关系，则a[i][j] = true，但是等价关系通常是隐式。添加一个关系并不是指将一个数组元素之为true。例如，一个等价关系定义在一个五个元素{a1,a2,a3,a4,a5}的集合上。该集合产生25对元素，每一对不是有关系就是没有关系。然而，a1～a2,a3～a4,a1～a5和a4～a2的信息隐含了所有的元素对都是有关系的。

等价类：

等价关系的另一种表示方法是采用等价类。集合S中的元素x的等价类是S的一个子集，它包含了所有与x有关的元素。这些等价类形成了不相交的集合 。等价类形成了S的一种分割

* + - 1. 不相交集合

等价类形成了集合S的一种分割。对于任意两个等价类Si和Sj，Si∩Sj=Φ，将所有的等价类并起来就是集合S。

这样的集合也称为不相交集合。

* + - 1. 不相交集合类的基本操作

find操作：找出特定元素属于哪个等价类）

union操作：用于添加关系。如果要把序偶(a, b)加到关系表中，则与a相关的元素都与b相关，与b相关的元素也都与a相关。即a的等价类与b的等价类合并为一个等价类。

* + - 1. 不相交集的实现

每个等价类表示为一棵树，等价类的名字为根结点的名字。这棵树不一定是二叉树。整个集合是一片森林

因为每个节点只需要知道父节点，可以采用双亲表示法，用一个数组保存。数组s[i]的值为i的父节点的下标。如s[i]=-1，表示i是某棵树的根。

为了执行两个集合的union操作，我们归并这两棵树，将一棵树的根作为另一棵树的孩子。这个操作很明显是常量时间。

元素x的find操作返回包含x的树的根。完成该操作的时间正比于从x到根的路径上的结点数。最坏情况可能是O(N)。

* + - 1. 改进的算法

尽管不相交集的union操作性能很好，而find操作的性能不佳。但find操作的性能问题是由union操作引起的。是union操作实现时没有关心所构造的树的结构。

改进的思想：尽量避免树的增高，可以按规模并，可以按高度并。

不相交集中，每棵子树的最理想的状态是一棵二层的数，只有根结点和它的儿子。这时，find操作的效率最高。

改进的union算法可以降低树的高度，提高find的效率。但当被归并的两棵树规模相同或高度相同时，树高还是会增加。

另外一种效率的改进方法是通过find操作，这种方法称为路径压缩。

当对find（x）采用路径压缩方法的话，那么在从x到根结点的路径上的每一个结点都将自己的父结点改为根结点。

如果x是一个很深的结点，这一改进将会大大降低树的高度 。

路径压缩能与按规模归并完美地兼容。路径压缩与按高度归并不完全兼容。

1. 实验项目
   1. 分析
      1. 而按高度求并不完全与路径压缩兼容，因为路径压缩会改变树的高度，而计算新的高度并不容易。怎么办呢？做法就是不计算，仍旧存储没有实行路径压缩之前的高度。这个高度并不是树的真实高度，而是一个估计高度，称为秩。
      2. 由于这个秩只是高度的估计值，所以我们可以直接借鉴改进过的union算法和改进过的find算法直接实现这个不相交集的类。
   2. 方案
      1. 构造不相交集类。
      2. 实现改进过的find算法和union算法。
      3. 测试不相交集类的运行的性能。
   3. 测试数据

int main()

{

int arr[15]{ -2,0,0,0,-2,4,-2,6,6,-4,9,9,11,12,12 };

DisjointSet disjoint\_set{ 15, arr };

cout << disjoint\_set.Find(7) << endl;

cout << disjoint\_set.Find(14) << endl;

disjoint\_set.Union(6, 9);

cout << disjoint\_set.Find(7) << endl;

cout << disjoint\_set.Find(14) << endl;

return 0;

}

* 1. 出错情况

未出错

* 1. 最终结果

6

9

9

9

**三、实验小结**

1. 重点

不相交集类的实现。

1. 难点

如何提高不相交集类的find函数和union函数的性能，同时保证整个不相交集类的函数的空间复杂度并不增加。

如何同时将按秩（高度）并和路径压缩进行整合来提高整个不相交集类的性能。

1. 收获、体会

数据存放的结构有很多种，每种数据结构都有自己的特性，在不相交集中，我们没有必要使用二叉树，而且如果使用了二叉树反而给归并函数造成了实现上的麻烦，利用适合和数据结构可以给我们进行实际问题的实现带来很大的方便。