**·讲稿**

我来为大家剖析斐波那契堆的结构。

斐波那契堆是一种松散的二项堆，与二项堆的主要区别在于构成斐波那契堆得树可以不是二项树，并且这些树的根排列是无序的（二项堆的根结点排序从左到右是按照结点个数排序的，不是按照根结点的大小）。

下面的这个图表现了斐波那契堆的样子。

这个表列出了不同的树实现插入、删除等基本操作的时间复杂度。很明显的可以看出，斐波那契堆在实现插入、删除、找到最小值、合并等操作的时间复杂度是O(1)的，这个性能在这些树中是很优越的。

这是斐波那契堆的数据结构，它分为斐波那契堆的节点的类与斐波那契堆的类。节点类中存储了关键字(键值)、度数、左兄弟、右兄弟、第一个孩子节点、父节点、是否被删除第1个孩子(1表示删除，0表示未删除)。

下面这张图表示了斐波那契堆在计算中的存储形态。可以看出，斐波那契堆可看作把多棵树的根节点直接用指针连接在一起而成。

对于斐波那契堆，取出最小值是其重要的进本操作，因此，我将具体剖析它的过程。

这是斐波那契堆取出最小值的一个例子。我们先来看看插入后的一些情况。

步骤一中，我们找到了最小值6，于是我们将6所在的树从堆中移除

步骤二中，我们将6取出了，然后把它的子节点直接连到斐波那契堆上

步骤三中，我们将斐波那契堆中的第一棵树移出，变成一个新的斐波那契堆。

从步骤四到步骤十，我们逐渐将老的斐波那契堆中的每棵树逐渐移到心得斐波那契堆

步骤十一中，合并新斐波那契堆中度相等的树。

这样取出最小值的操作就完成了。

下面是它具体的代码。

最后，我们来看应用。

题目是：找出图上一点到其它点之间的最短距离。

问题分析：这就是一个最普通的求最短距离的问题。在离散数学中，我们引进学习了Dijkstra来求解这个问题，那现在我们用什么数据结构来与这个算法结合呢？

下面就是Dijkstra算法的算法步骤

1、初始化结点

2、插入结点

3、从最小队列中，抽取最小点工作

4、松弛操作。

下面的伪代码具体描述了这个算法步骤的实现。并给出了每一步的时间复杂度。

这其中的relax过程是遍历一遍看通过刚刚找到的距离最短的点作为中转站会不会更近，如果更近了就更新距离，这样把所有的点找遍之后就存下了起点到其他所有点的最短距离。时间复杂度为O(1).

因此用斐波那契堆实现Dijkstra算法时间复杂度为O（V\*lgV + E），V为顶点，E为边数。

我给出了用其他数据结构，比如二项堆等实现Dijkstra算法的时间复杂度，它们均比用斐波那契堆要高，因此此题选用斐波那契堆。

我的讲解完毕，谢谢大家。