浅析树的划分问题

东北育才学校　贝小辉

【摘要】

树的最大－最小划分问题可以表述为如下形式：给定一棵 n 个节点的树以及每个节点的一个非负权值，要求将这棵树划分为 k 棵子树，使得子树中所有节点权值和的最小值最大。将原问题转化为对于给定下界，划分最多子树的问题，并通过对新问题的解决结合二分法来解决原问题是可行的，但是算法的总复杂度要依赖于节点权值的范围。本文接下来介绍了一个时间复杂度不依赖于节点权值范围的算法，随后通过对算法的描述、正确性的证明来进一步探讨算法的特点，并介绍了算法的一些扩展，最后总结了两个算法各自的特点。

【关键字】

问题转化，割，移动，上方

【正文】

树的划分问题是图论中的一类范围非常广泛的问题，这类题目的大意就是将给定的一棵树划分为若干棵子树，使其能够满足一定的条件或是使得某个特定的函数达到最值。如今，这类问题已被扩展出了各式各样的问题，并在很多领域都有着很广泛的应用。本文所要着重探讨的，是其中一种被称为最大－最小划分的问题。

一、　问题的提出

草莓（NOI2003 Day 2-2 berry　test6~test9）

题目大意：

给出一片草莓中每个草莓的重量以及它们的连接情况。定义：sum(i)表示第 i 块草莓田中所有草莓重量的和（1<= i <= k）,x = min{ sum(i) | 1<= i <=k }。你的任务就是要把一块草莓田分割成 k 块，且分割方案需要满足如下的条件：

- 每一块中的草莓必然是直接或者间接的和其他草莓相连接的；
- 这种分割方案所对应的 x 尽可能的大。

最后输出你的分割方案和结果。

这是一道提交答案式的题目，其中第 6 个数据至第 9 个数据所给的图是一棵树，若不考虑具体的数据情况，我们可以将原问题抽象成如下问题：给定一棵树以及树中每个顶点的一个非负权值，将树划分为 k 棵子树，定义：sum(i)表示第 i 棵子树中所有节点权值的和，x = min{ sum(i) | 1<= i <=k }，请求出 x 的最大值并输出此时的划分方案。

简单的说，就是将一棵树划分成 k 棵子树，使得其中权值和最小的那棵子树最大，我们把它称作最大－最小划分问题。

二、　算法 1：转化问题

1．转化为新问题

初次面对这个问题，或许我们会觉得无从下手，动态规划，贪心等经典算法在这里似乎都不适用，重要的是，我们不知道这个最小值是多少，令我们的思维过程遇到了很大困难。这时，我们不妨先考虑这样一个问题：

对于一个确定的下界，最多可将树划分为多少棵子树使得每棵子树的权值和都不小于此下界？

2．解决新问题

如果可以解决这个问题，我们便可以再通过二分法找到原问题的解。而事实上，新问题的解决只需要一个以贪心思想为基础的扫描算法。

> **扫描算法：**
>
> **按照深度由大至小的顺序扫描整棵树，对于扫描到的每个节点，计算以此节点为根的完全子树的权值和，若权值和大于等于给定下界，则将以这个节点为根的子树划分出来，并将其从原树中删去。直至整棵树所剩部分的权值和小于下界，将剩余部分归入最后划分出去的子树中。最终所得到的即是一种划分数目最多的最优划分。**

算法正确性的证明比较简单，这里略去了。容易看出，算法的时间复杂度是线性的，已经到达了理论的下界。接下来的任务就很简单了：我们可以通过二分法来找到最大的下界 x 使得划分的最大子树数目不小于 k，x 既为原问题的解。

3．小结

我们终于找到了一种解决问题的途径，但问题是否已经被完美的解决了呢？当我们分析这个算法总的时间复杂度是时候，我们发现它是依赖于节点权值的范围的。更加确切的说，如果每个节点权值的上界是 c，那么算法的时间复杂度就是 $O(n\log(nc))$。虽然在大多数情况下，程序的实际运行效率是比较好的，在解决 berry 那个问题时，所有数据也都能很快出解，但是当节点的权值范围很大或权值是实数时，算法便不是那么令人满意了。于是，我们自然而然地想到：能不能找到一个时间复杂度不依赖于节点权值范围的算法呢？

三、　算法 2：移动算法

1．新思路

在某一种划分中，如果一条边所连接的两个顶点属于两个不同的子树，那么我们就称在这条边上有一个“割”，我们注意到，每一个割都是对应着一棵子树的，就是这个割下方的所有顶点去掉其他割下方的顶点后所剩下的子树，而只有根节点所在的子树没有与之对应的割。于是，问题就可以转化如何为将 k-1 个割分配到 k-1 条边上，使其满足我们期待的最优条件。这样，我们就把问题的重点由对点的划分转化为对割的分配上，换了一种思维方式，我们希望能够因此而找到一种新的解决问题的途径。

我们还要介绍一种被称作“移动”的技术，一次移动被定义为将一个割从一条边移到

另一条与它相邻的边上，并且保证新的边一定是在原来那条边的下一层。也就是说，移动总是由上至下的，这样，我们就有可能通过一个给定的初始划分状态和一系列有限次的移动最后达到某个目标状态。初始状态的选择并不困难，我们可以任取一个度为 1 的顶点为根，然后在初始时将所有的割都放到那唯一一条与根节点相连的边上。这样，我们便有可能由初始状态达到任何一个我们想要的目标状态。问题的关键是，我们该如何制定每次移动的规则，使得其最后终止时会达到我们期望的最小子树最大的要求。

2．引入算法

而实际上，规则的制定远比我们想象的要简单的多，我们依据的还是一种贪心的思想：在进行每一步时，考虑所有可能的移动，并计算出每一种移动后在新位置的割所对应的新子树的权值和，我们取出其中新权值和最大的那一种，与当前未移动时的子树的最小权值和进行比较，若不小于当前的最小值，则进行这步移动，否则算法就结束了。具体算法流程如下：

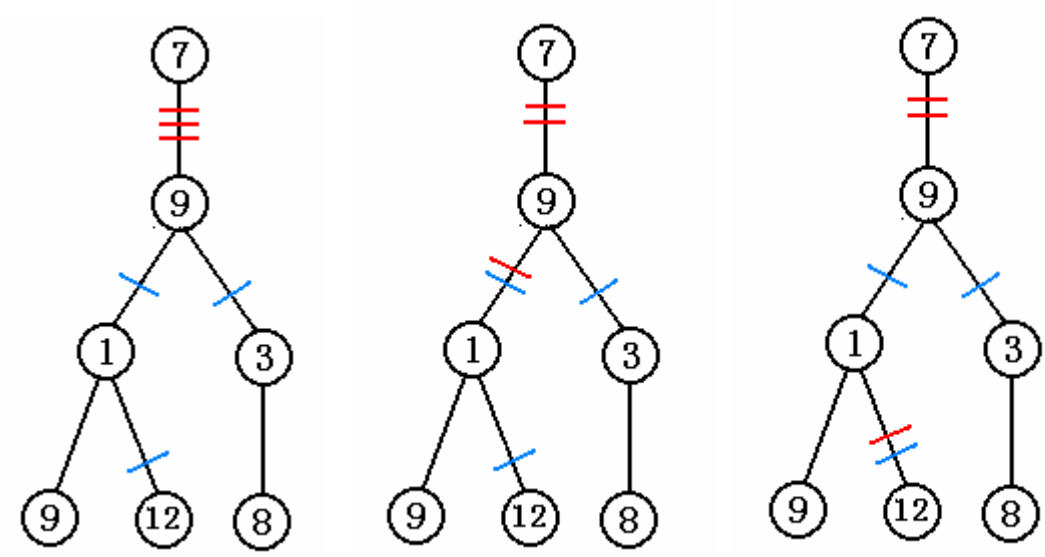
移动算法：

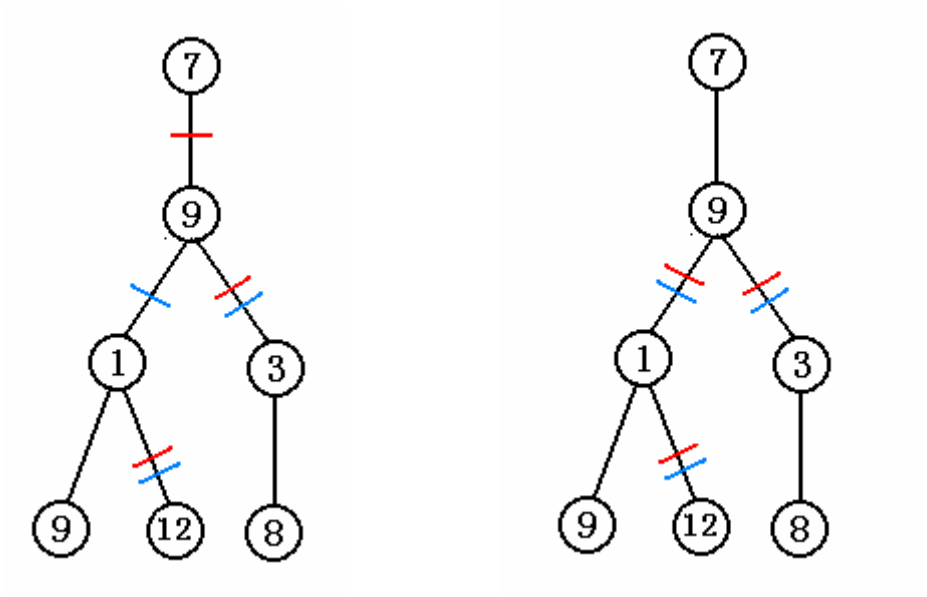
1. **选择任何一个度为 1 的顶点作为树的根，将 k-1 个割都放在与根相连的唯一的一条边上。**
2. **计算出当前划分状态下的子树权值和的最小值 Wmin。**
3. **考虑所有可能的移动，找出能使移动后的割所对应的子树权值和 Wnow 最大的那种移动。**
4. **如果 Wnow ≥ Wmin，那么进行这步移动，并转到步骤 2。**
5. **算法结束，Wmin 既为所求的最大的最小值，当前划分即为一种最优划分。**

当然，我们还需要证明这个算法的正确性，毕竟这个算法用到了很多的新东西，我们无法在第一时间内用直觉来确定它是对的，我们还需要对它进行更多的研究和分析。

3．定义“上方”

在证明算法之前，我们常常要用几个简单的例子来观察它的实现过程，这次也不例外。





在观察了上图以及其他一些简单的例子后，我们发现了一个很有趣的事实：对于算法进行的每一步移动后，产生的新的划分状态总是在某个最优划分的“上方”，而每次操作都是将当前的划分“向下”移动一些，最后直到和某个最优划分重合为止。这就为我们的证明提供了一个很不错的思路：我们可以定义在划分之间的一种“上方”的关系，然后证明算法进行中的每一种划分状态都是在某个最优划分的上方，而当一种划分是在某个最优划分上方的时候，算法一定会继续。这样，我们就证明了原算法的正确性。

于是，我们便试图对划分之间的“上方”关系做一个定义。自然而然的想法就是：划分 A 在划分 A’的上方，也就是存在一种 A 的割和 A’的割的一一对应，使得每个 A 的割都在它所对应的 A’的割的上方。这种定义能够形象的表现出上方的含义，不失为一种不错的定义方法，但为了在证明中更好的应用“上方”这种关系，我们还希望能够找到它的更加实用的性质。

在这之前，让我们先定义一下部分子树的概念：若一棵 T 的子树 T’包含了顶点 v 连同 v 的某一个儿子以及这个儿子的所有后继，则称 T’是 T 在顶点 v 处的一棵部分子树。与 v 相连的唯一一条边被称为 T’的初始边。

接下来我们考察在树 T 上的两个划分 A 和 A’，若 A 在 A’上方，则对于任意一棵 T 的部分子树，我们发现，若 A 中有一个割 c 在子树上，因为这个割所对应的 A’中的割 c’是在 c 的下方，所以 c’也一定在这棵部分子树上。如果将划分 A 在一棵部分子树上的割的数目表示为# (A)。我们得到了如下性质：若划分 A 在 A’上方，则对于树 T 的任意一棵部分子树，都有# (A) ≤ # (A’)。这条性质在下面的证明过程中起到了非常重要的作用。

下面我们便来到了最核心的部分：为了证明算法的正确性，我们将试图证明如下几点：

4．算法的证明

(1) 在初始状态时的划分 **A** 是在任何一个最优划分 **Q** 的上方的。

(2) 若存在一个最优划分 Q 使得当前的划分 A 是在 Q 的上方，且 A 和 Q 不相等，则算

法一定不会终止。

(3) 设 A 在 Q 的上方且 A 不等于 Q，在算法进行一步后 A 变为 A’，我们一定还能找到一个最优划分 Q’使得 A’在 Q’上方。

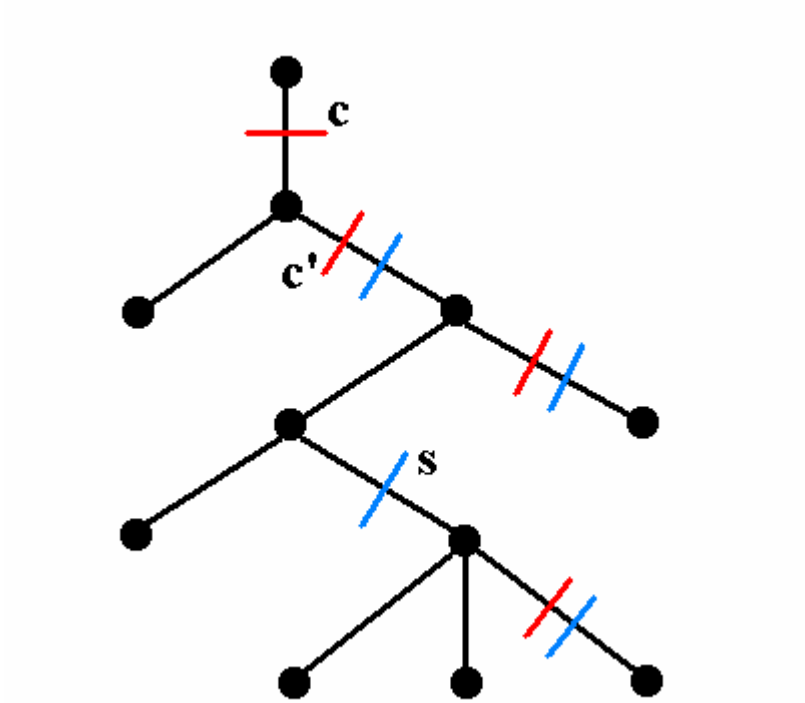
(4) 算法会在有限步内终止，算法终止时的划分一定是一个最优划分。

(1)的正确性是很显然的。(4)也很容易说明：首先，每步移动都是由上至下的，这些割不会被无限移动下去，故算法会在有限步内终止。若算法终止时不是最优划分，由(3)得一定存在一个最优划分 Q 使得当前划分 A 是在 Q 的上方，又因为 A 和 Q 不相等（A 不是最优划分），由(2)得算法一定还会继续，与算法终止矛盾，故算法终止时的划分一定为最优划分。下面我们来证明(2)和(3)的正确性。

在本文中，字母 A 通常表示由算法进行而得到的划分，字母 Q 表示一个最优划分，即使得最小子树最大的划分。用 Wmin(A)表示在划分 A 下的最小子树的权值，则对于任意一个划分 A，都有 Wmin(A) ≤ Wmin(Q)。

*(2) 若存在一个最优划分 **Q** 使得当前的划分 **A** 是在 **Q** 的上方，且 **A** 和 **Q** 不相等，则算法一定不会终止。*

证：因为 A 和 Q 不相等，故在 A 中必存在一个割使得在同一条边上没有 Q 的割，令 c 为满足此条件的深度最大的割，由于在以 c 所在的边为初始边的部分子树上，# (A) ≤ # (Q)，且 c 所在的边上没有 Q 的割，所以在子树中必存在一个 Q 的割 s 使得在同一条边上没有 A 的割，取 s 上方遇到的第一个 A 中的割 c’，将 c’向 s 的方向移动一步，则 c’所对应的新子树的权值和 ≥ s 所对应子树的权值和 ≥ Wmin(Q) ≥ Wmin(A) （因为 Q 是最优划分），所以算法一定会继续，且移动后的割所对应的新子树的权值和一定不小于 Wmin(Q)。证毕



上面的证明还顺便得出了另一个结论：每次经过算法移动后的割所对应的新子树的权值和一定不小于 **Wmin(Q)**，这个结论我们将在**(3)**的证明中再次用到。

(3) 设 A 在 Q 的上方且 A 不等于 Q，在算法进行一步后 A 变为 A’，我们一定还能找到一个最优划分 Q’使得 A’在 Q’上方。

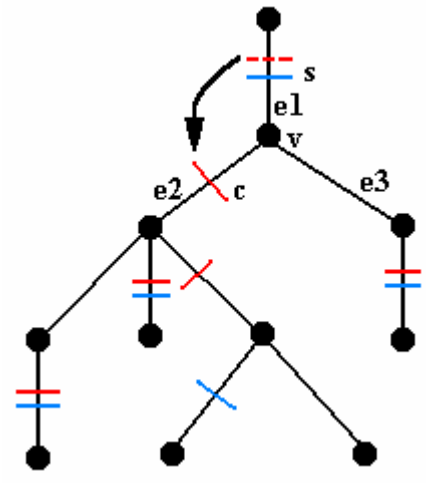
证：我们不妨假设算法进行的移动是将一个割 c 从边 e1 移动到了 e2，设边 e1 与 e2 交与点 v。为了构造 Q’,我们分以下两种情况进行讨论：

情况 1：对于在顶点 v 处的每一棵部分子树 T’，都有# (A)=# (Q)。

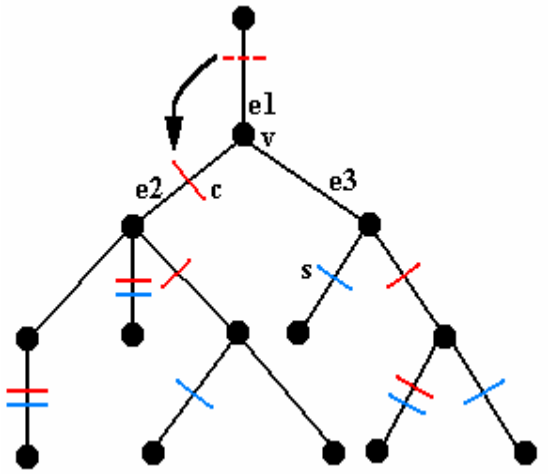
则经过移动后对于以 e2 为初始边的部分子树有# (A’) = # (Q)+1，由于在以 e1 为初始边的部分子树中# (A) ≤ # (Q)，故在边 e1 上必有 Q 的一个割，不妨设为 s，将 s 从 e1 移动到 e2，形成 Q’，这样在以 e2 为初始边的部分子树 T’中仍有# (A’) ≤ # (Q’)，故 A’仍是在 Q’上方的。所以在 T’中，s 所对应的子树一定是包含 c 所对应的子树的。所以 s 所对应子树的权值和 ≥ c 对应子树的权值和 ≥ Wmin(Q) ((2)的证明中的得出的结论)，所以 Q’也是一个最优划分，且 A’是在 Q’上方的。

情况 2：存在在顶点 v 处的某棵子树 T’使得# (A) < # (Q)。

如果在以 e2 为初始边的子树中就有# (A) < # (Q)，则 A’≥ Q，既 Q 就是符合条件的划分。若在以 e2 为初始边的子树中# (A) = # (Q)，则由假设我们可以设存在一条从 v 出发的边 e3 使得在以 e3 为初始边的部分子树 T’中# (A) < # (Q)，设 s 为 Q 在 T’中深度最小的割，将 s 移至 e2 处，构成划分 Q’。则 A’是在 Q’的上方的。与情况 1 一样，s 所对应的新子树的权值和一定大于等于 Wmin(Q)，而对于顶点 v 所在的子树，由于它现在包含的 s 原来对应的子树，故它也一定是大于等于 Wmin(Q)的，由于由 Q 变至 Q’所改变的只有这两棵子树的情况，故 Q’也是一个最优划分，且 A’在 Q’上方。　　证毕



情况 1



情况 2

至此，算法的正确性已经被证明完毕。回顾整个证明过程，“上方”的概念自始至终都起着非常重要的作用，而实际上，“上方”的概念就是一种序的关系。我们通过引入“上方”的概念将状态间看似杂乱无章的关系变得有序化，有条理性，从而解决问题，这在我们信息学竞赛中是很值得借鉴的。

5．时间复杂度

我们再来关注一下算法的时间复杂度，虽然算法的描述并不复杂，但在实际操作中我们会发现，若只是单纯的按照算法的流程去做时间效率是很低的，我们需要对算法进行一些必要的优化使得它能够高效的完成流程中的每一步操作。经过多年的研究，这个算法目前已知的比较好的时间复杂度是 $O(k^2 rd(T)+kn)$，其中 rd(T)是这棵树的半径，具体的算法实现比较复杂，这里就不详细讲了，有兴趣的朋友可以参考一下相关的论文。

6．算法的扩展

算法终于得到了，但是我们并不满足，我们还希望能够将算法扩展一下，使它的适用范围更加广泛。首先，让我们来看一下权函数方面。本题中，每棵子树的权被定义为子树中所有节点的权之和。我们不由得想到，对于一些其他的权函数，比如一棵子树中节点权的最大值，子树的直径等等，这个算法是不是依然适用呢？我们发现，在算法正确性的证明过程中，只有（2）和（3）的证明用到了子树的权函数，而通过更加深入的观察发现，证明中只是利用了权函数的一个性质：若 T'是 T 的任意一棵子树，则必然有 W(T) ≥ W(T')，其中 W(T)表示树 T 的权函数。也就是说，只要权函数满足这个条件，那么这个证明就是正确的，这个算法也就是可行的。于是，对于满足特定条件的一类问题我们都找到了一种通用的解法，这也显示出这个算法很强的可扩展性。

有了对权函数的扩展，我们接下来要便想到了对于问题的扩展。例如将树划分为 k 棵子树，使得子树的最大值最小，或是最大值与最小值的差最小等等问题，我们是不是也可以用原算法求解呢？事实证明，单纯的照搬是行不通的，因为算法有它自身的特点，不可能适用于所有的问题，但解决问题的思路却是我们可以借鉴的。在解决与这一类型有关的问题时，我们或许可以改变移动的规则，或修改一下“上方”的定义，从而设计出符合题目特点的算法，但这些已不在本文的讨论范围之内了。

7．用算法 1 优化算法 2

最后，我们还要提一提算法 2 的优化。在算法 2 中，初始状态是将 k－1 个割都放在与根相连的唯一一条边上得到的，这样做的好处是一定可以保证它是在某个最优划分的上方的。可是由这个初始状态移动到最优状态往往需要很多步的移动，那么有没有更好的初始状态呢？我们想到了本文开头所介绍的扫描算法，我们发现，在扫描算法中，如果把下界设为（树中所有节点的权值和）/ (k-1)，则算法划分出来的子树数目一定不超过 k-1，我们将剩余的割都放在与根相连的那条边上，可以证明，这个划分状态一定是在某个最优划分的上方的，而由它达到最优状态所需要移动的步数却减少了，于是，只需要一个线性的预处理，我们便得到了一种更好的初始状态。这样，算法 1 和算法 2 便巧妙的结合在一起了。

四、　总结

本文介绍了解决树的最大－最小划分问题的两种算法，它们通过不同的方式思考问题，从而得到了不同的解法。算法 1 主要应用了问题转化的思想，将原问题转化为新的，容易解决的问题，再通过对新问题的求解以及二分法解决原问题，算法实现简单，时间效率也不错。而算法 2 则从另一个角度看问题，将目光集中再分割子树的“割”上面，从而得到了一个复杂度不依赖与节点权值范围的算法，在算法证明过程中起了很大作用的划分间“上方”的关系实际上是一种“序”的思想，这些思想在信息学竞赛中都有着非常广泛的应用。

在如今的信息学竞赛中，试题越来越灵活多变，往往无法再用单纯的经典算法来解决问题。这就要求我们更多的思考，只有在对问题进行了深入的思考以后，才能更加深刻地了解到问题的本质，从而设计出符合题目特点的优秀的算法来。

【参考文献】

[1] Ronald I.Becker and Yehoshua Perl, The shifting algorithm technique for the partitioning of trees, *J.ACM* 28 (1981) 5-15

[2] Yehoshua Perl and Stephen R.Schach, Max-min tree partitioning, *Discrete Applied Mathematics* 62 (1995) 15-34