题意：给你 n 个数 a[i],每次选 l~r 堆合并果子，问最小代价

另一种题意：建立一颗叶子节点数目为 n，叶权值为 a[i]，每个非叶节点的孩子数目在 l~r 之间的多叉最小哈弗曼树。

std 提出的贪心：使用[l-1,r-1]之间的数，找一个关于 n-1 的长度最小且字典序最小的拆分数序列，按照这个序列做合并果子（建哈弗曼树）。

为了证明 std 的正确性，需要逐步证明以下三点

① 如果在合并 k 次下有解，那么合并 k 次下的最优解一定优于合并 k+1 次下的最优解。

② 在①的前提下，也就是假定合并 k 次下存在最优解，那么这个合并的操作序列越不均匀（大的数越大，小的越小），解越优。

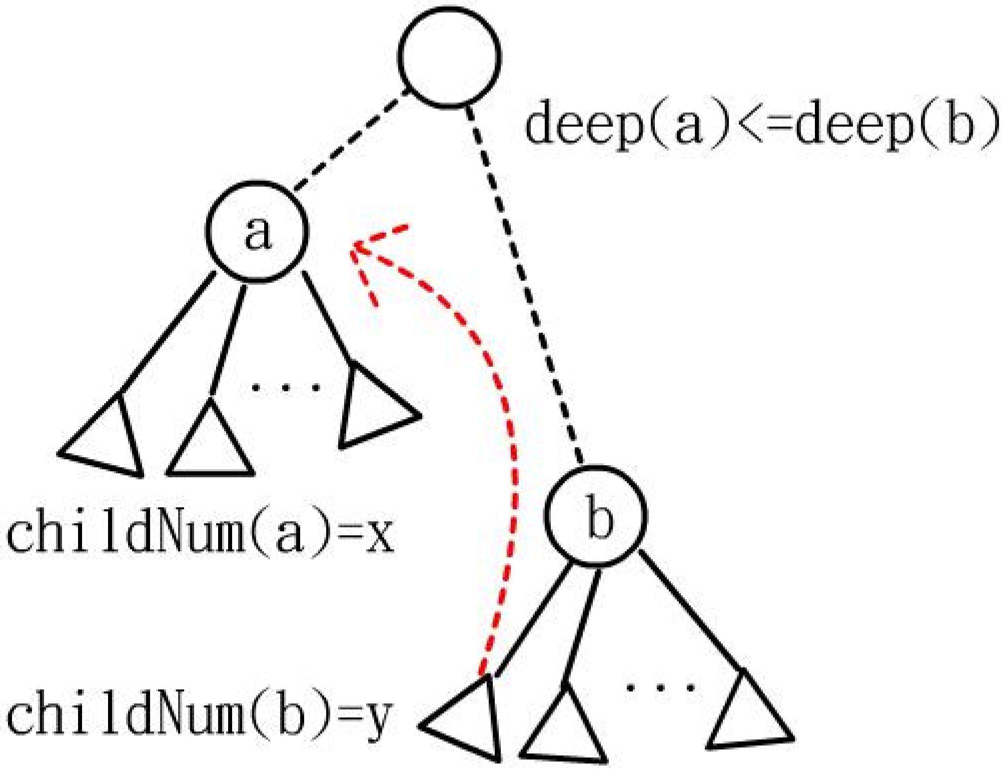
③ 在①②的前提下，也就是假定合并 k 次下存在最优解，且假定这 k 个数的分布最不均匀，那么按照合并的操作序列升序排列时得到最优解。

首先证明①，这个我认为是显然的，因为如果进行 k 次操作就足以将所有堆合并成一堆，那么进行 k+1 次合并成一堆就必然使得一些堆进行了本可以省去额外的合并操作。

（也许有更好的说法，待补）

接下来证明②的正确性。

如果结论①正确，那么就保证了我建立的哈弗曼树的叶节点数目为 n，非叶节点数位 k，总节点数目为 n+k。



如图，deep()表示节点的深度，childNum()表示节点的直接后代数目。
当哈夫曼树的节点数目固定时，假设树上存在两非叶节点 a,b 且 deep(a)<=deep(b)时，假设 a 的孩子数目为 x， b 的孩子数目为 y。
那么只要 x≠r 且 y≠1，那么这个树结构就一定不如把 b 节点的一些孩子接到 a 后面更优。

我们发现此调整是当且仅当 x≠r 且 y≠1 时就要进行的，与树结构无关，因为树上要么 deep(a)<=deep(b)，要么 deep(a)>=deep(b)。不是 a 给 b 就是 b 给 a，那么最终小的越小，大的越大，结果越优。

（其实取等号的时候是不影响的，如果两节点在哈弗曼树上的 deep 相同那其实接在谁后面都一样）

那么在进行完若干轮调整后，先不管树结构如何，非叶节点的 childNum 数组就首先被确定下来了。

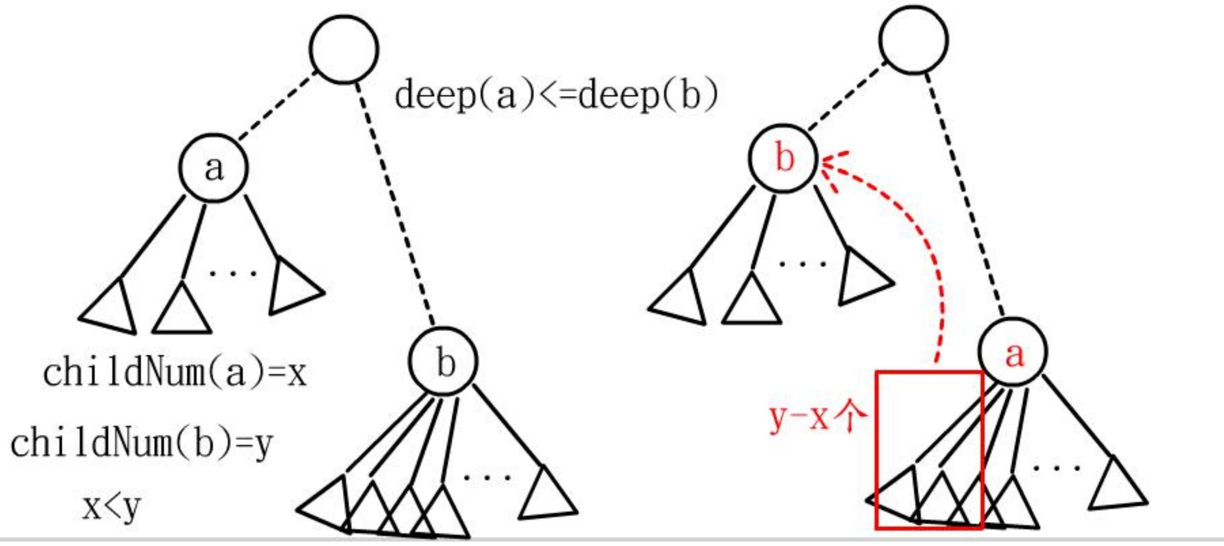
那么至此证明了②的正确性。

;①确定了哈弗曼树的节点数目，②确定了非叶节点的孩子数目。

接下来证明③的正确性。

定理一：在节点数目确定，非叶节点的孩子数目确定的多叉哈弗曼树中，如果 childNum(a)>childNum(b)，则必有 deep(a)<=deep(b)是该树结构成为最优哈弗曼树的**必要条件。**

定理的证明:



反证法：

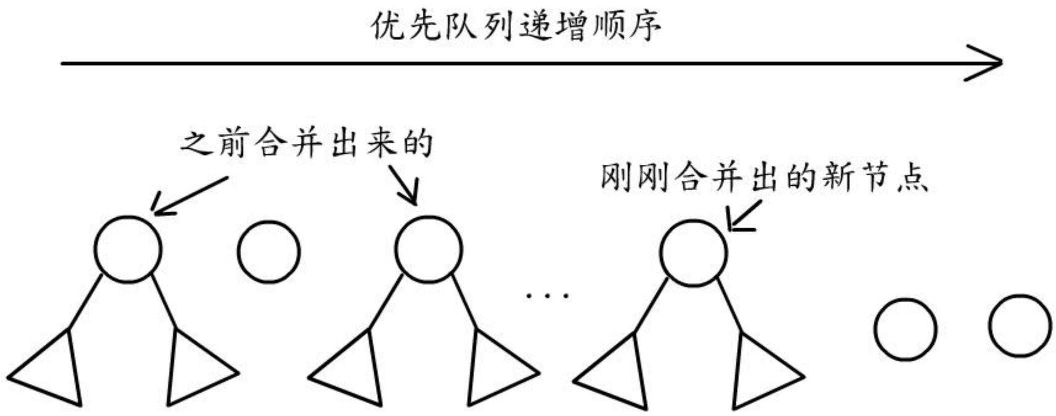
现在节点 a 的孩子数目确定为 x（不可更改），b 的孩子数目也确定为 y(不可更改)。

假设存在 x<y 且 deep(a)<deep(b)的情况，那么该树结构一定不如把 a 放到 b 的位置，把 b 放到 a 的位置，然后将原来 b 孩子中的 y-x 个孩子跟着 b 一起走更优。

既然 childNum(a)>childNum(b)，则必有 deep(a)<=deep(b)是该树结构成为最优哈弗曼树的**必要条件。**

那么只要保证这个的前提下做建立哈弗曼树的合并果子操作，建出来的树就是最优哈弗曼树。

定理二：假设在**二叉哈弗曼树**中，第 i 个非叶节点建立的时间戳为 ti,则对于两个非叶节点 i,j 若存在 ti<tj 则必有 deepi>=deepj。



节点在哈夫曼树上的深度也可被解释为该节点被创建后参与合并的次数。
显然在二叉哈夫曼树中，刚刚合并出的新节点总是当前所有靠合并产生的新节点中权值最大的那个。（换句话说就是哈弗曼树在建立的过程中新节点的权值单调递增）
那么当 ti<tj 时，节点 j 每参与一次合并，i 因为总是在合并后排在 j 的前面，那么 i 必然进行一次合并。也就是当 deepj+1 时 deepi 总是在这之前就+1 了。

定理二的引理：

假设在**多叉哈夫曼树中，若新建节点的孩子数目单调非降**，第 i 个非叶节点建立的时间戳为 ti,则对于两个非叶节点 i,j 若存在 ti<tj 则必有 deepi>=deepj。

证明同定理二，只要保证哈弗曼树在建立的过程中新节点的权值单调递增则此结论就是正确的。

结合定理一以及定理二的引理，如果按照升序的顺序建多叉哈弗曼树。

那么则有当 ti<tj 时，childNum(i)<childNum(j)并且此时 deepi>=deepj。那么按照此顺序建立的多叉哈弗曼树就是最优哈弗曼树。

至此证明③的正确性。

总的来说就是我需要这三步分别确定三个东西

① 确定了哈弗曼树的节点数目

② 确定了非叶节点的孩子数目

③ 确定了哈弗曼树的建立顺序。

那么这三步之后就确定了最优解就是找一个关于 n-1 的长度最小且字典序最小的拆分数序列，按照这个序列做合并果子。