Introduction to Algorithm
ID: 519021911248
Name: ZhuoHao Li

Homework 3

Question 1.

1. 板材切割问题：给定一块非常长的钢板（不考虑宽度），要求在长度方向上将其切割为 N 小块，长度分别为 $L_1, L_2, \cdots, L_N$。未切割前钢板长度为切割后各小块长度之和。每次切割操作所需要的开销为当前被切割的板材长度。比如给定一块长度 10 的钢板，需要将其切割为长度分别为 2, 3, 5 的小块，第一次将其切割为 5, 5 两小块，付出开销 10，第二次将长度为 5 的小块切割为长度为 2, 3 的两小块，付出的开销为 5，则总共付出的开销为 15。请求出按照题目要求将钢板切割完的最小开销是多少？需要给出优化子结构和贪心选择性质，给出伪代码，并分析算法的复杂度。
注：可参考霍夫曼编码的思路，将切割过程等效为树型结构，N 个小块作为 N 个叶节点，将切割总开销建模为叶节点长度的多项式。

My Answer 1. 这道题和 *Huffman code tree* 本质上是相同的。将切割过程等效为树形模型，N 个被切割好的小块作为 N 个叶节点，求解的过程实际上就是构造"代价"最小的二叉式，将切割总开销建模为叶节点长度的多项式. 事实上，我们可以直接把每个小块的长度与板材的总长的比值视作其"频率"，最优解一定是尽可能先取"频率"最小的小块，这符合贪心算法的思想，具体的证明和 *Huffman code tree* 相似，更详细证明见后文。
具体的做法如下：
1. 用一个集合 *set* 存储所有被切割成的 N 块小块.
2. 选取 *set* 中长度最小的两块，不妨设为 $L_i, L_j, i \neq j$，并作为叶子结点，将 $L_i + L_j$ 的值作为新的小块 L_k 的长度，把 L_i, L_j 从 *set* 中删除，并且将 L_k 添加到 *set* 中.
3. 重复上面的计算，直到 *set* 里只有一个节点为止（事实上，这个节点就是 *root* 节点）
具体实现可以参考 *Huffman code algorithm:*
1. 用一个 *Heap* 或者 *PriorityQueue* 用来存储小块（*Heap or PriorityQueue* 会根据小块的长度进行重构）
2. 只要 *Heap or PriorityQueue* 不空，连续两次 *pop* 出堆顶/队列头的节点，这两个节点长度一定是最小的两个，将这两个节点的长度相加构建新的节点，并把原来 *pop* 出的两个节点作为新节点的两个 *child*，*push* 新节点入堆/队.
3. 重复操作！
伪代码如下所示：（采用 *Heap*）

Introduction to Algorithm
ID: 519021911248
Name: ZhuoHao Li

Homework 3

Algorithm 1 MINCOST

Require: $n \geq 0$

```
1:  initialize a Set called S
2:  n = S.size
3:  Q = S                                        ▷ 以堆来实现相关操作
4:  MINLENGTH ← 0
5:  for i ←1 to n-1 do
6:      z=Allocate-Node().                       ▷ 构造空的新节点
7:      z.left=x=EXTRACT-MINLENGTH(Q)            ▷ 堆操作
8:      z.right=y=EXTRACT-MINLENGTH(Q)           ▷ 堆操作
9:      z.length=x.length+y.length               ▷ 新节点赋值
10:     MINLENGTH+=z.length
11:     INSERT(Q,z)                              ▷ 新节点入堆，随后堆进行重构
12: end for
13: return MINLENGTH
```

复杂度分析:

Q 是一个 *HEAP*，第 *2* 步用堆排序的 *BUILD-HEAP* 构造 *HEAP:O(n);* 后续每一次堆重构都会带来 *O(logn)* 复杂度，循环 *n-1* 次需要 *O(nlogn)*，所以最终的时间复杂度为：$T(n)=O(n)+O(nlogn)$，当 $n \to \infty$, $T(n)=O(nlogn)$.

正确性证明：

1. 贪心选择性质：

给定一棵分割树，分割树的根节点长度为所有小块的长度总和，树每一层代表了这一次的分割操作，一棵分割树就对应一个分割方案。经过简单的数学计算，一个分割方案的所需代价如下计算：

$$B(T) = \sum_{s \in S} s.length \times d_T(s)$$

上式中 s 表示 S 中的某一个小块，这些小块都会出现在生成分割树的叶子结点上，$s.length$ 代表该小块的长度，$d_T(s)$ 代表该小块在树中的深度。

（引理 *1*）设 S 是所有被切割而成的小块，$\forall s \in S$, s 的长度用 $s.length$ 来表示。设 x, y 是 S 中长度最小的两个子块，则存在一个 S 的优化分割树 T，使得 x, y 在 T 中是深度最大的兄弟叶子结点，即 x, y 在最后一步才被分割。

（引理 *1*）证明：

证明思路：令 T 为一棵已有最优分割树，然后对其进行变换，得到另外一棵最优分割树，使在新树中，x 和 y 是深度最大叶结点，且它们是兄弟结点。如可以构造一棵这样的分割树，那么 x 和 y 将在最后一步才被分割。设 T 中 a 和 b 是两个深度最大的兄弟结点，不失一般性，设 $a.length \leq b.length$ 且 $x.length \leq y.length$. 因为 x,y 是被认为长度最小的两个节点，所以必然有 $a.length \geq x.length$, $b.length \geq y.length$.(在下面的证明中，认为 $x.length \neq b.length$，否则将有 $a.length=b.length=x.length=y.length$, 此时引理 *1* 显然成立)。在 T 中交换 x 和 a 生成新树 T'，然后交换 T' 中的 y 和 b 生成新树 T'', 则在 T'' 中 x,y 变为两个深度最深的叶节点。T 和 T' 的代价差为：

$$B(T) - B(T') = \sum s.length \times d_T(s) - \sum s.length \times d'_T(s)$$

$$= x.length \times d_T(x) + a.length \times d_T(a) - x.length \times d'_T(x) - a.length \times d_T(a)$$

$$= x.length \times d_T(x) + a.length \times d_T(a) - x.length \times d_T(a) - a.length \times d_T(x)$$

$$= (a.length - x.length)(d_T(a) - d_T(x)) \geq 0$$

由此可知，交换 x 和 a 不会增加代价，同理，交换 y 和 b 不会增加代价, 因此有 *B(T") ≤ B(T') ≤ B(T);* 又由于 T 是最优的，即 *B(T) ≤ B(T")*，因此，*T"* 也是最优的分割树，且 *x,y* 是其中深度最深的兄弟叶节点，引理成立。

2. 优化子结构：
*(*引理 *2)* 设 T 是 S 的最优化分割树，$\forall\ s \in S$, *s.length* 是 s 的长度。设 *x,y* 是 T 中长度最小的两个节点，则他们必然为两个相邻的叶子节点，z 是他们的父节点，则 z 的长度认为是 *z.length=x.length+y.length* 的新生成节点，从而 *T'=T-x,y* 是 *S'=(S-x,y)* $\bigcup$ z 的最优分割树。

*(*引理 *2)* 证明：

$$B(T) = B(T') + x.length + y.length:$$

$$\forall v \in S - x, y, d_T(v) = d'_T(v), so:$$

$$v.length \times d_T(v) = v.length \times d_T(v')$$

$$cause\ \ d_T(x) = d_T(y) = d'_T(z) + 1, so$$

$$x.length \times d_T(x) + y.length \times d_T(y)$$

$$= (x.length + y.length)(d'_T(x) + 1) = (x.length + y.length)d'_T(x) + x.length + y.length$$

$$z.length = x.length + y.length, so$$

$$x.length \times d_T(x) + y.length \times d_T(y) = z.length \times d'_T(z) + x.length + y.length$$

$$so,\ \ B(T) = B(T') + f(x) + f(y)$$

假设 *T'* 不是 *S'* 的最优分割树，则必然存在另一棵树 *T"* 是 *S'* 的最优分割树，*B(T")<B(T')*，因为 *z∈C'*，所以按照算法的生成规则，z 一定作为 *T"* 中的叶子结点存在，重新将 *x,y* 加入到假设的树 *T"* 中并且作为 z 的子节点构成一棵新树 *T"'*，则存在 S 如下形式的分割树 *T"'*, 满足：

$$B(T''') = ... + (x.length + y.length)(d''_T(z) + 1) =$$

$$...... + (x.length + y.length) \times d''_T(z) + x.length + y.length$$

$$= B(T'') + x.length + y.length < B(T') + x.length + y.length = B(T)$$

必然可以通过 *T'* 构造一棵更优的分割树，这与 T 是最优分割树矛盾！所以 *T'* 必然是 *C'* 的最优分割树！

综上，此算法正确性得证。