hw9 due 11.18

> 17.1-2；17.3-1， 17.3-4；19.1-3(in text v2)；19.2-2(in text v2)， 19.2-6(in text v2)；

17.1-2

17.1-2 证明：如果 k 位计数器的例子中允许 DECREMENT 操作，那么 n 个操作的运行时间可能达到 $\Theta(nk)$。

假设输入是一个1后面跟着 $k-1$ 个零。如果调用DECREMENT操作，必须更改 k 个条目。如果之后对这个结果调用INCREMENT操作，它将撤销这 k 次更改。因此，通过交替调用它们 n 次，总的时间复杂度是 $\Theta(nk)$。

17.3-1

17.3-1 假定有势函数 Φ，对所有 i 满足 $\Phi(D_i)\geqslant\Phi(D_0)$，但 $\Phi(D_0)\neq 0$。证明：存在势函数 Φ'，使得 $\Phi'(D_0)=0$，对所有 $i\geqslant 1$ 满足 $\Phi'(D_i)\geqslant 0$，且使用 Φ' 的摊还代价与使用 Φ 的摊还代价相同。

定义 $\Phi'(D)=\Phi(D)-\Phi\left(D_0\right)$ 。那么，如果有$\Phi(D)\geq\Phi\left(D_0\right)$ ，则可以得出 $\Phi'(D)=\Phi(D)-\Phi\left(D_0\right)\geq\Phi\left(D_0\right)-\Phi\left(D_0\right)=0$ 。并且 ，$\Phi'\left(D_0\right)=\Phi\left(D_0\right)-\Phi\left(D_0\right)=0$ 。最后，使用 Φ' 的摊还成本是 $c_i+\Phi'\left(D_i\right)-\Phi'\left(D_{i-1}\right)=c_i+\left(\Phi\left(D_i\right)-\Phi\left(D_0\right)\right)-\left(\Phi\left(D_{i-1}\right)-\Phi\left(D_0\right)\right)=c_i+\Phi\left(D_i\right)-\Phi\left(D_{i-1}\right)$ ，这与使用 Φ 的摊还成本相同。

17.3-4

17.3-4 执行 n 个 PUSH、POP 和 MULTIPOP 栈操作的总代价是多少？假定初始时栈中包含 s_0 个对象，结束后包含 s_n 个对象。

由于 $D_n=s_n$ 和 $D_0=s_0$ ，且从空栈开始的 n 个栈操作的摊还成本是 $O(n)$，方程 17.3 暗示了摊还成本是 $O(n)+s_n-s_0$ 。

在这个上下文中，s_n 和 s_0 分别表示操作完成后和开始前的栈大小。因此，摊还成本不仅包括操作的基本成本（这里是 $O(n)$ ），还包括由栈大小变化导致的额外成本或节省。这意味着整体成本与操作数量成线性关系，同时还受到栈的初始和最终大小的影响。

19.1-3

19.1-3 如图 19-4 所示，假设按后序遍历顺序，将二项树 B_k 中的结点标为二进制形式。考虑深度 i 处标为 l 的一个结点 x，且设 $j=k-i$。证明：在 x 的二进制表示中共有 j 个 1。恰包含 j 个 1 的二进制 k 串共有多少？证明 x 的度数与 l 的二进制表示中，最右 0 的右边的 1 的个数相同。

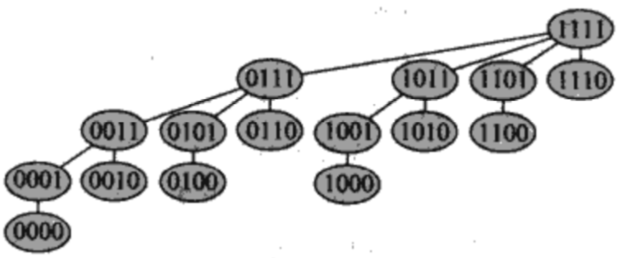

图 19-4 二项树 B_4，其各结点按后序遍历次序标以一个二进制数

19.2 对二项堆的操作

这个问题涉及到二项树 B_k 的一个特性，其中节点的二进制表示与其在树中的位置有着特定的关联。考虑二项树 B_k，它是一种特殊的树，其中每个节点都有一个特定的二进制标签。

1. **节点的二进制表示中含有 j 个 1：**
   - 在后序遍历顺序中，一个深度为 i 的节点 x 在二项树 B_k 中被标记为一个长度为 k 的二进制数，其中 $j=k-i$。
   - 在二项树中，深度 i 的节点意味着从根节点到该节点有 i 条边。
   - 每向下移动一层，意味着二进制表示中的一个 1 被转换为 0（因为节点的子节点比父节点在二进制表示中少一个 1）。
   - 因此，深度为 i 的节点在其二进制表示中将有 $j=k-i$ 个 1。
2. **恰含有 j 个 1 的二进制 k 串的数量：**
   - 求这个数量相当于从长度为 k 的二进制数中选择 j 个位置放置 1。
   - 这是一个组合问题，可以用组合数公式计算：$C(k, j)$，即从 k 个不同元素中选择 j 个元素的组合数。
3. **节点的度数与二进制表示中最右边 0 右侧 1 的数量相同：**
   - 在二项树 B_k 中，一个节点的子节点数（即度数）等于它在二进制表示中从右数起第一个 0 右侧的 1 的个数。
   - 例如，在上述树中，节点 0110（位于 B_4 中）的度数为 2，因为从右边数起第一个 0 的右边有两个 1（即 11）。
   - 这是因为每个 1 代表一个子节点，且子节点的数量由最低位开始向左数的连续 1 的数量决定。

通过以上分析，可以看出二项树的这些性质与节点的二进制表示密切相关。

19.2-6

19.2-6 假设无法表示出关键字 $-\infty$。重写 BINOMIAL-HEAP-DELETE 过程，使之在这种情况下能正确地工作。运行时间仍应为 $O(\lg n)$。

```
class BinomialHeapNode:
    def __init__(self, key):
        self.key = key
        self.parent = None
        self.child = None
        self.sibling = None
```

```python
        self.degree = 0

def binomial_link(y, z):
    y.parent = z
    y.sibling = z.child
    z.child = y
    z.degree += 1

def binomial_heap_union(H1, H2):
    # 合并H1和H2，返回新堆的头部
    # 实现细节省略

def binomial_heap_extract_min(H):
    # 提取并返回H中的最小元素
    # 实现细节省略

def binomial_heap_decrease_key(H, x, new_key):
    if new_key > x.key:
        raise ValueError("new key is greater than current key")
    x.key = new_key
    y = x
    z = y.parent
    while z is not None and y.key < z.key:
        y.key, z.key = z.key, y.key  # 交换键值
        y = z
        z = y.parent

def binomial_heap_delete(H, x):
    # 假设H是二项堆的头部，x是要删除的节点
    # 找到堆中的最小关键字
    min_key = float('inf')
    current = H
    while current is not None:
        if current.key < min_key:
            min_key = current.key
        current = current.sibling

    # 将x的关键字减小到小于最小关键字
    binomial_heap_decrease_key(H, x, min_key - 1)

    # 提取最小元素（现在是x）
    binomial_heap_extract_min(H)

# 以下是二项堆的其他必要部分，如创建堆、插入节点等，这里省略
```