PS6-231830106

Problem 1

(a):

由于 $m = 2^p - 1$，我们可以将 m 表示为 $m = 111..1_2$（二进制表示，有 p 个1）。当我们对 k 进行模 m 运算时，等价于将k中的每个元素的二进制相加后再模m，而如果x和y由相同的元素构成，则显然两者模m后都是元素二进制相加后模m的结果，因此显然x和y会hash到一些相同的值。

(b):

1. 线性探测（Linear Probing）

哈希表索引	0	1	2	3	4	5	6	7	8	9	10	1
值					4	10	15	17	22	28	31	5

2. 二次探测（Quadratic Probing）

哈希表索引	0	1	2	3	4	5	6	7	8	9	10	1
值					4	10	17	15	22	28	31	5

3. 双重哈希（Double Hashing）

哈希表索引	0	1	2	3	4	5	6	7	8	9	10	1
值					4	10	15	17	22	28	31	5

Problem 2

在U中所有元素组成的对数有$|U|(|U|-1)$个，又有$Pr[h(k)=h(l)] \leq \epsilon$，因此任意哈希函数的期望的碰撞数量必然小于等于$|U|(|U|-1)\epsilon$。又$Pr[h(k)=h(l)=b] = \frac{1}{|B|}(k, l \in U, b \in B)$，于

是有式：

$$Pr[h(k)=h(l)=b]\times|U|(|U|-1)\le|U|(|U|-1)\epsilon$$

解得$\epsilon\ge\frac{1}{|B|}$，显然有$\epsilon\ge\frac{1}{|B|}-\frac{1}{|U|}$，即证。

Problem 3

证明：
设无扩展的插入操作均摊成本为3，无最小的删除操作均摊成本为3。
数学归纳法：
Basis: 第一次操作前账户余额为0
Hypothesis: 第i次操作前账户余额不为0
Inductive Step：
考虑第i次操作共有四种情况
1. 无扩展的插入，此时账户余额+2，显然仍然大于0
2. 无缩减的删除，此时账户余额+3，显然仍然大于0
3. 有扩展的插入，假设是从n扩展到了2n：
1.假设上次大操作是扩展，则是从2/n到n的扩展，本次扩展时元素的个数为(3n/4)，而上次扩展时元素个数为(3n/8)，因此不包括删除(若有删除余额更多)，至少有(3n/8)次插入因此账户余额至少为(3n/4)，足够应对本次扩展的开销，因此i次操作后账户余额依然大于0
2.假设上次大操作是缩减，则是从2n到n到缩减，本次扩展时元素的个数为(3n/4)，而上次缩减时元素个数为(n/2)，因此不包括删除(若有删除余额更多)，至少有(n/4)次插入因此账户余额至少为(3n/4)，足够应对本次扩展的开销，因此i次操作后账户余额依然大于0
4. 有缩减的删除，假设是从n缩减到了n/2:
1.假设上次大操作是缩减，则是从2n到n的缩减，本次缩减时元素的个数为n/4，而上次缩减时元素个数为(2/n)，因此不包括插入(若有插入余额更多)，至少有(n/4)次删除因此账户余额至少为(3n/4)，足够应对本次缩减的开销，因此i次操作后账户余额依然大于0
2.假设上次大操作是扩展，则是从n/2到n的扩展，本次缩减时元素的个数为n/4，而上次扩展时元素个数为(3n/8)，因此不包括插入(若有插入余额更多)，至少有(n/8)次删除因此账户余额至少为(3n/8)，足够应对本次缩减的开销，因此i次操作后账户余额依然大于0
综上即证。

Problem 4

(a):

显然在n次调用INC中SOMETHINGELSE被调用n次则，其开销为4n，平均到每次操作中为O(1)级别的开销，因此均摊时间复杂度为O(1)

(b):

n次调用SOMETHINGELSE的时间开销为$\sum_{i=1}^{n}2^i=2^{n+1}-2$，平均到每次为$\frac{2^{n+1}-2}{n}$，因此均摊时间复杂度为$O(\frac{2^n}{n})$。

(c):

同理均摊时间复杂度为$O(\frac{4^n}{n})$。

Problem 5

设$x.size = N, x.left.size = L, x.right.size = R$，$T_x$为以$x$为根的子树

(a):

1. 思路：有$L+R+1=N$，且$L \le N/2$，$R \le N/2$，若N为奇数则L=R=(N-1)/2，若N为偶数则L=N/2，R=N/2-1（互换也行），由此可见一个1/2-balance tree即为一个平衡二叉搜索树。因此算法思路很简单，中序遍历T_x得到排序好的数组，再递归构建一个平衡二叉搜索树即可。
2. 伪代码

```
function InorderTraversal(BST, root, sortedArray)
    if root is NULL:
        return
    InorderTraversal(BST, root.left, sortedArray)
    sortedArray.append(root.value)
    InorderTraversal(BST, root.right, sortedArray)

function BuildBinaryBst(sortedArray,start,end)
        if start > end
        return NULL

    mid = (start + end) / 2
    root = NEW TreeNode(sortedArray[mid])
    root.left = BuildBalancedBST(sortedArray, start, mid - 1)
    root.right = BuildBalancedBST(sortedArray, mid + 1, end)
    return root

function Convert(BST, root)
    sortedArray = EMPTY ARRAY
    InorderTraversal(BST, root, sortedArray)
    root = BuildBalancedBST(sortedArray, 0, length(sortedArray) - 1)
    return root
```

(b):

不妨设$L \ge R$，且有$L+R+1=N, L \le \alpha \cdot N$，$R \le \alpha \cdot N$，设树高为H，则在最恶劣情况下有$N \times \alpha^H = 1$，解得$H = \lg_{1/\alpha} N$即，H = $O(\lg n)$，而树高就是搜索时间复杂度，因此search的时间复杂度为$O(\lg n)$

(c):

1. 显然在BST中，对于任意节点x，Δ(x)≥ 0，因此显然Φ(T)≥ 0，因此任何BST的势能都是非负的
2. 在(a)中分析可知，对于1/2-balance tree有$|L-R|\leq 1$，则显然Φ(T)=0。

(d):

在这个不平衡的大小为m的树中讨论，由于$\Delta(x)\leq(2\alpha-1)m+1$，要保持势能足以支付一个m势能的操作(rebuild)则要有$c\geq\frac{1}{2\alpha-1}$。此时在rebuild操作时，rebuild后的树为一个平衡二叉搜索树显然其势能为0，因此有$\Delta\Phi(T)=O(\Delta(x)\times c)=O(m)$，即证

(e):

由(d)可知$c\geq\frac{1}{2\alpha-1}$，在Insert和delete时，若不需要重建，则插入/删除的时间复杂度等价于search的时间复杂度，为$O(\lg n)$；若需要重建，插入/删除的新节点至多影响树高个节点的$\Delta(x)$，$\Delta\Phi(T)=O(\Delta(x)\times\lg n)=O(\lg n)$，势能差足以支付rebuild的开销，此时显然rebuild不改变整体的均摊时间复杂度，因此删除和插入的均摊时间复杂度为$O(\lg n)$。