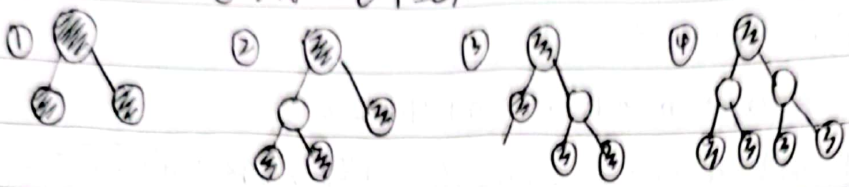
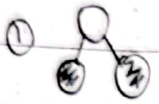
Problem Set 4 (ddl: 4.4)

(◉表示黑色节点, ○表示红色节点)　RB0: ◉

RB1:

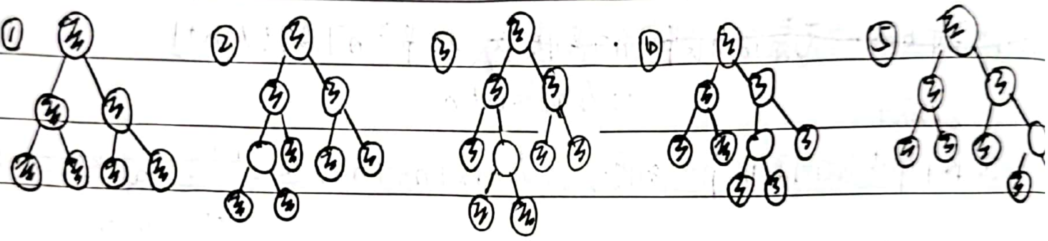


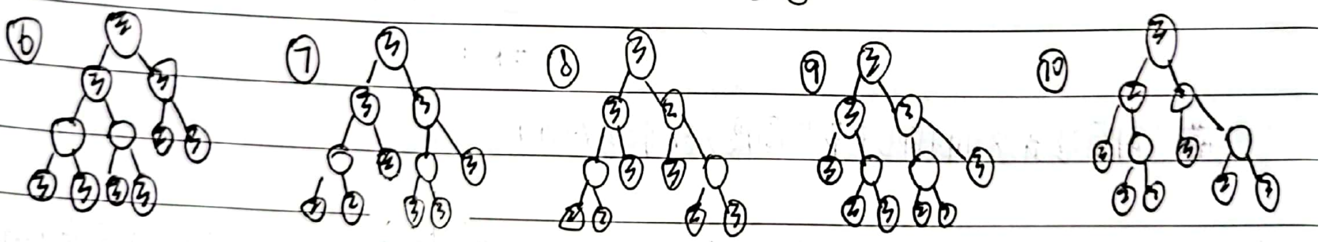
ARB1

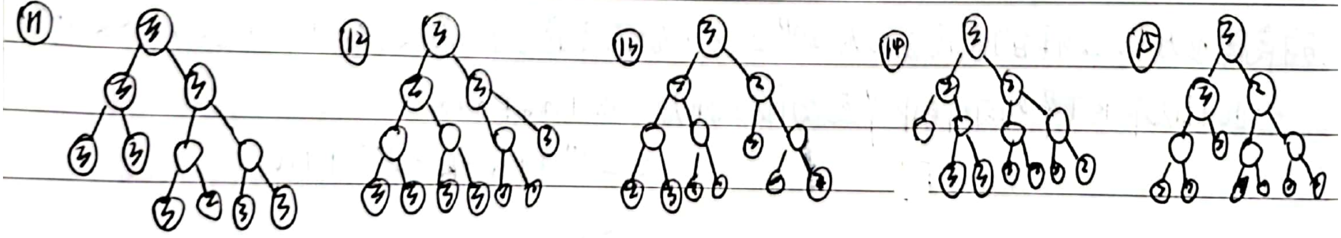


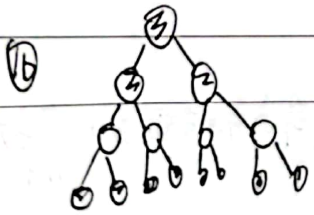
RB2的构造: 在RB1的①.②.③.④的基础上, 对每个外部黑色节点再添加其子节点, 其子节点为RB0 tree 或ARB1 tree

如①的构造: RB2 tree



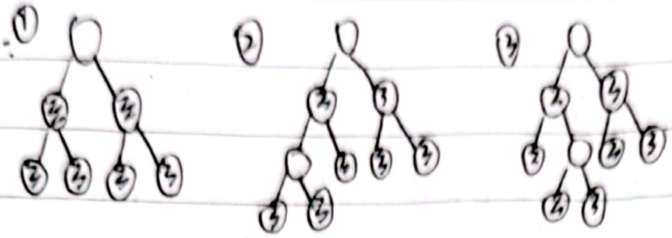






其余的三种也用相同的方法构造, 不再一一列举。

ARB2:



…… 多为种

构造方法：在ARB1的基础上对所有的外部节点添加子节点，其子节点为RB0 tree或ARB1 tree。

Problem 4.1.2 (BG 6.6)

(i)
Prove: Let T be an RB_h tree. That is, let T be a red-black tree with black height h. Then:

1. T has at least $2^h - 1$ internal black nodes.
2. T has at most $4^h - 1$ internal nodes.
3. The depth of any black node is at most twice its black depth.

(ii)
Let A be an ARB_h tree. That is, let A be an almost-red-black tree with black height h. Then:

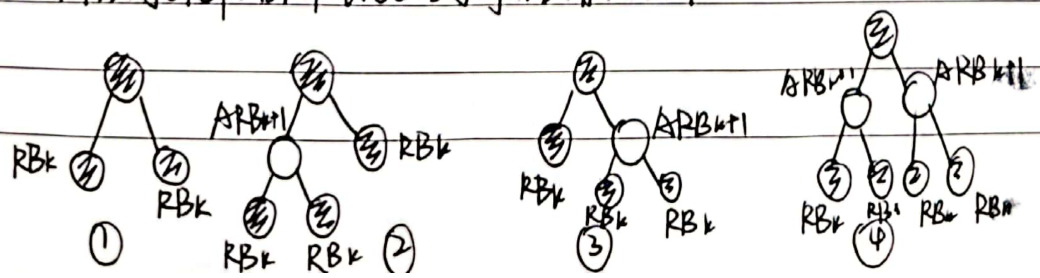
1. A has at least $2^h - 2$ internal black nodes.
2. A has at most $\frac{1}{2}(4^h) - 1$ internal nodes.
3. The depth of any black node is at most twice its black depth.

证明(i)：

h=0时，对于RB0 tree来说，有至少$2^0-1=0$个内部黑色节点，至多有4^0-1个内部节点，黑色深度为0，任意黑色节点的深度满足至少为其两倍，显然可见RB0 tree满足(i)。

假设∃k>0且k∈Z，对于∀h∈[0, k]，RBh tree都满足(i)。

h=k+1时，考虑RBk+1 tree的左右子树，由RBh tree的定义可知，RBk+1 tree的子树要么是RBk tree，要么是ARBk+1 tree，而ARBk+1 tree的子树只会是RBk tree，所以可画出RBk+1 tree的可能情形如下：



对于情形①：

其左右子树 $h\in[0,k]$，∴由归纳假设可知其左右子树的内部黑色节点至少有 $(2^{k}-1)\times2=2^{k+1}-2$ 个，则①中的内部黑色节点至少为 $2^{k+1}-2+1=2^{k+1}-1$ 个，满足(引)中1。

又由归纳假设可得其左右子树的节点总数至多为 $(4^{k}-1)\times2$，则①中的内部节点总数至多为 $(4^{k}-1)\times2+1=2^{k+1}-1\le4^{k+1}-1$，满足(引)中2。

又由归纳假设可得左右子树为根节点时任意黑色节点的深度至多为其黑色深度的两倍，即 $Dnode\le2Black(Node)$，则新增根节点后任意黑色节点深度为 $Dnode+1$，其黑色深度为 $Black(Node)+1$，由于 $Dnode\le2Black(Node)$，

$\therefore Dnode+1\le2(Black(Node)+1)$，∴(引)中3得证。

对于情形②、③：

对比情形①多了1个RBk，所以内部黑色节点数也至少为 $2^{k+1}-1$，满足(引)中1。

三个RBk的节点总数至多为 $(4^{k}-1)\times3$，再加1个红色节点和1个根节点可得节点总数至多为 $3\times4^{k}-1\le4^{k+1}-1$，∴满足(引)中2。

对于根的左右子树中为RBk的一侧由情形①可知(引)中3成立，对于ARBk+1的一侧来说，在原先RBk的基础上增加了两个根节点，其黑色深度为 $Black(Node)+1$，深度为Dnode相当于+2，满足 $Dnode+2\le2(Black(Node)+1)$，∴(引)中3得证。

对于情形④：

对比情形①多了2个RBk，所以内部黑色节点数至少为 $2^{k+1}-1$，满足(引)中1。

四个RBk的节点总数至多为 $(4^{k}-1)\times4$，再加上2个红色节点和1个根节点可得节点总数至多为 $4\times4^{k}-1=4^{k+1}-1\le4^{k+1}-1$，∴满足(引)中2。

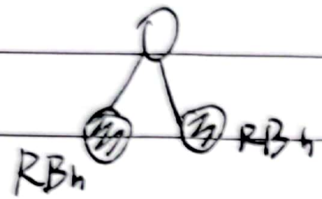
由情形①和情形②、③中对(引)中3的分析可知情形④也满足(引)中3。

综上所述，(引)成立。

……有$(2^{k}-1)\times 2=2^{k+1}-2$个内部……

证明(iii)

对于ARBh tree来说，它的情形仅有1种：



由(ii)可知2个RBh tree的内部黑色节点至少有$(2^{h}-1)\times 2=2^{h+1}-2\geq 2^{h+1}-2$，∴(iii)中1成立

又由(i)可知2个RBh tree最多有$(4^{h}-1)\times 2$个内部节点，ARBh tree再加上1个红色根节点最多有$(4^{h}-1)\times 2+1=\frac{1}{2}\times 4^{h}-1=\boxed{\frac{1}{2}}4^{h}-1$个内部节点，(iii)中2由(i)中证明情形②.③的正……可知成立，∴(iii)成立。

Problem Set 4.2

10 mod 10　(10+2)

Problem 4.2.1

Given a hash table with m=11 entries and the following hash function h_1 and step function h_2:

$h_1(key)$=key mod m; $h_2(key) = \{key\ mod(m-1)\}+1$.　(10+1) mod 11 = 0

6 0 9 2 6 8 3 10 10

Insert the keys {72, 11, 42, 68, 6, 30, 47, 98, 10} in the given order (from left to right) to the hash table using each of the following hash methods:

1.Close address hash with $h_1 \Rightarrow$h(k) = h_1(k)　(6+21) mod 11 = 5

2.Linear-Probing with $h_1 \Rightarrow$ h(k,i) = (h_1(k)+i) mod m

3.Double-Probing with h_1 as the hash function and h_2 as the rehashing function $\Rightarrow$ h(k,i) = (h_1(k) + ih_2(k)) mod m　(6+7) mod 11 = 2　(6+14) mod 11 = 9

1.

0 → 11
1
2 → 68
3 → 47
4
5
6 → 6 → 72
7
8 → 30
9 → 42
10 → 10 → 98

2.

0	1	2	3	4	5	6	7	8	9	10
11	10	68	47			72	6	30	42	98

3.

0	1	2	3	4	5	6	7	8	9	10
11	10	68	47		6	72		30	42	98

Problem 4.2.2 (BG 6.19)

The type of a hash table H under closed addressing is an array of list references, and under open addressing is an array of keys. Assume a key requires one "word" of memory and a linked list node requires two words, one for the key and one for a list reference. Consider each of these load factors for closed addressing: 0.25, 0.5, 1.0, 2.0. Let h_c be the number of hash cells in the hash table for closed addressing.

a. Estimate the total space requirement, including space for lists, under closed addressing, and then, assuming that the same amount of space is used for an open addressing hash table, what are the corresponding load factors under open addressing?

b. Now assume that a key takes four words and a list node is five words (four for the key and one for the reference to the rest of the list), and repeat part (a).

a.

① $\alpha = 0.25 = \frac{n}{m}$ $\therefore n = 0.25hc$ 所占空间 $= 0.25hc \times 2 + hc = 1.5hc$

$\alpha_{open} = \frac{0.25hc}{1.5hc} = \frac{1}{6}$

② $\alpha = 0.5 = \frac{n}{m}$ $\therefore n = 0.5hc$ 所占空间 $= 0.5hc \times 2 + hc = 2hc$

$\alpha_{open} = \frac{0.5hc}{2hc} = \frac{1}{4}$

③ $\alpha = 1.0 = \frac{n}{m}$ $\therefore n = hc$ 所占空间: $hc + hc \times 2 = 3hc$

$\alpha_{open} = \frac{hc}{3hc} = \frac{1}{3}$

④ $\alpha = 2.0 = \frac{n}{m}$ $\therefore n = 2hc$ 所占空间: $hc + 2hc \times 2 = 5hc$

$\alpha_{open} = \frac{2hc}{5hc} = \frac{2}{5}$

b. ① $\alpha = 0.25 = \frac{n}{m}$ $n = 0.25hc$ 所占空间: $0.25hc \times 5 + hc = 2.25hc$

$\alpha_{open} = \frac{0.25hc}{2.25hc \div 4} = \frac{4}{9}$

② $\alpha = 0.5 = \frac{n}{m}$ $n = 0.5hc$ 所占空间 $= 0.5hc \times 5 + hc = 3.5hc$

$\alpha_{open} = \frac{0.5hc}{3.5hc \div 4} = \frac{4}{7}$

③ $\alpha = 1.0 = \frac{n}{m}$, $n = hc$, 所占空间: $5hc + hc = 6hc$ $\alpha_{open} = \frac{hc}{6hc \div 4} = \frac{2}{3}$

④ $\alpha = \frac{n}{m} = 2.0$　$n = 2m = 2hc$　∴所占空间为 $2hc \times 5 + hc = 11hc$

$\alpha_{open} = \frac{2hc}{11hc \div 4} = \frac{8}{11}$

Problem 4.2.3

考虑一个负载因子是 α 的开放寻址哈希表，请找一个 α 值，使得一次不成功查找的预期探测次数是一次成功查找预期探测次数的 2 倍。

一次不成功查找的预期次数：$\frac{1}{1-\alpha}$，一次成功查找的预期次数：$\frac{1}{\alpha}\ln\frac{1}{1-\alpha}$

则 $\frac{1}{1-\alpha} = \frac{2}{\alpha}\ln\frac{1}{1-\alpha}$

$\alpha \approx 0.7153$

Problem 4.2.4 (Huang 18.5)

给定多重集合 S（集合中相同取值的元素可以多次出现）。集合 S 支持两个操作：

- INSERT(S, x)：将元素 x 插入到 S 中。插入操作支持元素的重复插入，所以集合 S 中的元素不一定是唯一的。
- DEL-LARGER-HALF(S)：将集合 S 中最大的 $\lceil \frac{|S|}{2} \rceil$ 个元素删掉。

请给出上述操作的算法实现，并证明操作序列的平摊代价为 $O(1)$.

~~使用~~ 算法实现：插入集合S时直接插入末端并计数，删除元素时首先使用最坏情况线性时间选元素择算法找出第 $\lceil \frac{|S|}{2} \rceil$ 大的元素，然后将比它大的元素删除即可 上述

对算法的操作用记账法进行分析：插入一个元素的实际代价 C_{act} 1，删掉 $\lceil \frac{|S|}{2} \rceil$ 个元素的实际代价为 $k|S|$ ($k|S| \in O(|S|)$)，设插入操作的记账代价 $C_{acc} = k$，删除操作的记账代价 C_{act} 为 $-k \cdot |S|$，由于删除的元素首先会被插入进S中，所以插入操作预先为删除操作中的搜索、遍历、删除代价预支了cost，所以 $C_{acc插} + C_{acc删} \geq 0$，又∵ $C_{act插} + C_{acc插} = k+1$，$C_{act删} + C_{acc删} = 0$　∴单次操作的平摊代价为 $O(k+1) = O(1)$

Problem Set 4.3

Problem 4.3.1 (BG 6.23)

Exhibit a *Union-Find* program of size n which requires $\Theta(n \log n)$ time using the straightforword find (without path compression) and the weighted union (wUnion).

weight

伪代码实现：

Find

w Union

```
struct Node{
        int key=-1;
        int weight=0;
Node Parent[n];
int Find(int index){
    while(parent[index].key!=-1)
        index=parent[index].key;
    return index;}
void w-Union(int x1, int x2){
    int p1=Find(x1), p2=Find(x2);
    Node n1=Parent[p1], n2=Parent[p2];
    if(p1!=p2)
    { if(n1.weight>n2.weight)
        { n1.weight+=n2.weight,
          parent[p2].key=p1;
        }
      else{
          n2.weight+=n1.weight, Parent[p1].key
          =p2;
      }
}}
```

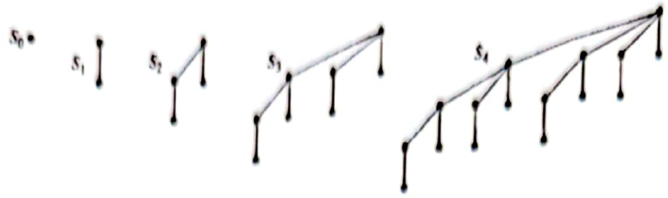


图 1: Binomial trees, also called S_k trees

Problem 4.3.2 (BG 6.25)

Binomial trees, also called S_k trees, are defined as follows: S_0 is a tree with one node. For $k > 0$, an S_k tree is obtained from two disjoint S_{k-1} tree by attaching the root of one to the root of the other. See Figure. 1 for examples.

Prove that, (i) if T is an S_k tree, T has 2^k vertices, height k, and a unique vertex at depth k. The node at depth k is called the handle of the S_k tree.

证明: k=0时，S_0 tree 有1个顶点（$2^0=1$），高度为0，在深度为0的地方只有一个节点，符合(i)。

假设k=n-1 (n≥1)时，S_k tree也满足(i).

则 k=n 时，S_n tree 由两个 S_{n-1} tree 连接而成，所以 S_n tree 有 $2^{n-1}\times 2=2^n$ 个顶点，一个S_{n-1} tree 的根节点成为了另一个 S_{n-1} tree 的子节点，由于 S_{n-1} tree 的高为 n-1，所以根节点的子节点的最大高度为n-1，所以 S_n tree 的高度为 n，同时可知 S_n tree 中以子节点为根的树中节点仅有一个深度为 n-1，其余深度均小于 n-1，因此 S_n tree 中以根的 仅有一个深度为n的节点。因(i)得证。

Problem 4.3.3

对于一组变量 $x_1, x_2, ..., x_n$，给定一些形如 $x_i = x_j$ 的等式约束和形如 $x_i \neq x_j$ 的不等式约束，这些约束是否能同时满足？例如，如下一组约束

$$x_1 = x_2, x_2 = x_3, x_3 = x_4, x_1 \neq x_4$$

是无法同时满足的。请给出一个高效算法，判断关于 n 个变量的 m 个约束是否可以同时满足。

使用加权并(WEIGHTED-UNION)和路径压缩"查"(C-Find)来解决该问题。

先把所有等号连接的式子作为输入，将等号连接的两个变量使用(WEIGHTED-UNION)建立联系，然后再将不等号连接的式子作为输入，通过C-Find查找两个元素是否在同一个集合中，若在一个集合中则无法满足，否则可以满足，继续判断其余不等式。

Problem 4.3.4

(**Ternary Counter**) We now use an array A of length k (starting from 0, 1, ..., to $k-1$) as a ternary counter, that is, the value in $A[i]$ could only be 0, 1, or 2. Initially, the value of the counter is, of course, 0. The only allowable operation is *increment*(A), which adds 1 to the current number in the array. The following table shows how the elements in A changes when we apply four increment operations. The cost of an operation is the number of elements that change. When the value of the counter is n (that is, we have applied n increment operations), what is the amortized cost of an increment operation.

operation	...	$A[2]$	$A[1]$	$A[0]$	cost of this operation
increment(A)	...	0	0	1	1
increment(A)	...	0	0	2	1
increment(A)	...	0	1	0	2
increment(A)	...	0	1	1	1

操作	Cact	Cacc	Camo
无进位增1	1	1	2
有进位增1	k+1	-k+1	2

说明: 对A[i]中某一位的变化作分析，变化的情况只有3种(0→1, 1→2, 2→0)，这3种情况的分析均是一致的，若是无进位的增1，表示只有该位发生了这3种变化中的一种，若是有进位的增1，则表示前k位(当前位的前面元素)发生了2→0的变化该位发生了3种中的一种，所以可以按有进位和无进位来分析.

将无进位增1的Cacc设为1，有进位增1的Cacc设为-k+1，在有进位增1时前面无进位增1以及自身的-k+1中预支的+1已经按前为k做了积攒 ∴ Cacc有 + Cacc无 ≥ 0

∴ 单次操作的平摊代价不超过2. counter计数到n时的总代价 ≤ 2n ∈ O(n)