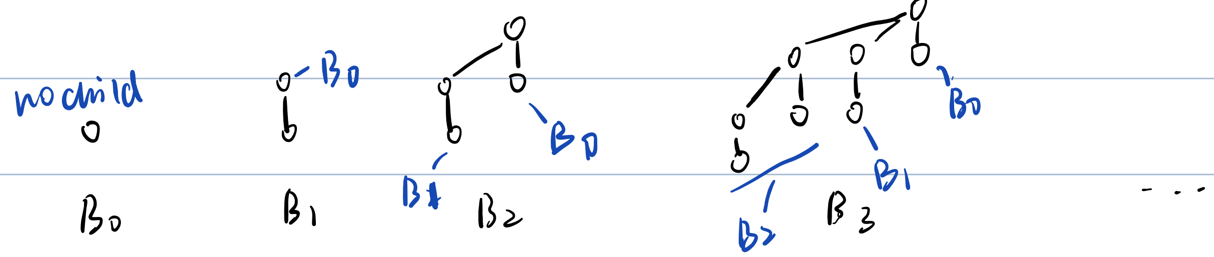
Binomial Queue.

是由一些 binomial tree 组成的 forest → 二项树

Binomial tree 递归定义

height 0 : o one node tree

height k: B_k 为 B_{k-1} attach to the root of B_{k-1}.



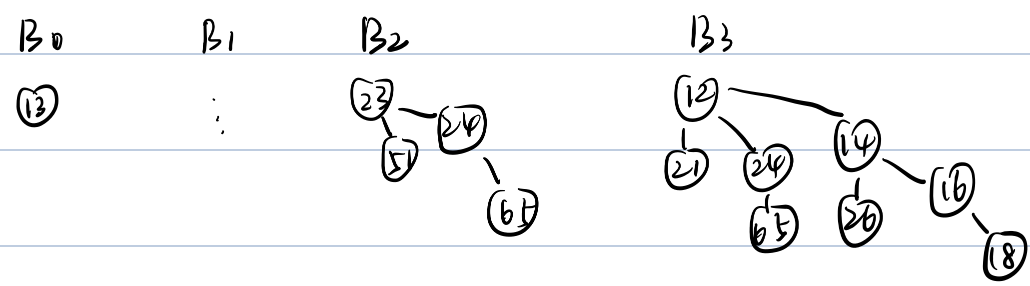
B_k 有 k 个儿子, $B_0 \cdots B_{k-1}$, 有 $2^0+2^1+\cdots+2^{k-1} = 2^k$ nodes. the number of nodes at depth d is C_k^d → 二项式系数.

Binomial queue

一个优先队列 heap 都可以被一群 Binomial tree 唯一表示.

eg: 一个 size 13 的 heap:

$13 = \overset{3210}{1101}_2$ → no B_1



Operations

FindMin ⇒ 对每个树检查各个 root.

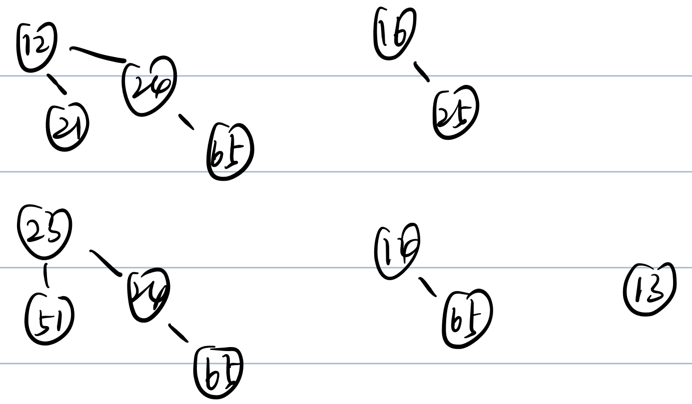
N nodes 至多有 $\lceil \log_2 N \rceil$ 个节点, $T_p = O(\log N)$

我们记录 min 然后去更新它, O(1)

merge

H1

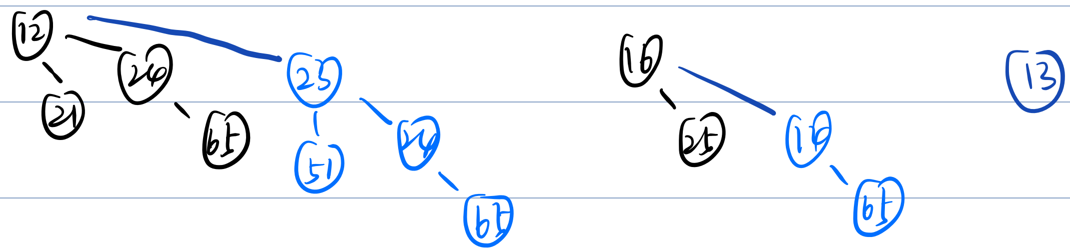
H2



=> 110

$$\begin{array}{r} 110 \\ +\ 111 \\ \hline 1101 \end{array}$$

=> 111



Sort by height

$T_p = O(\log N)$

insert

eg. insert 1, 2, 3, 4, 5, 6, 7 into an empty queue.

1.

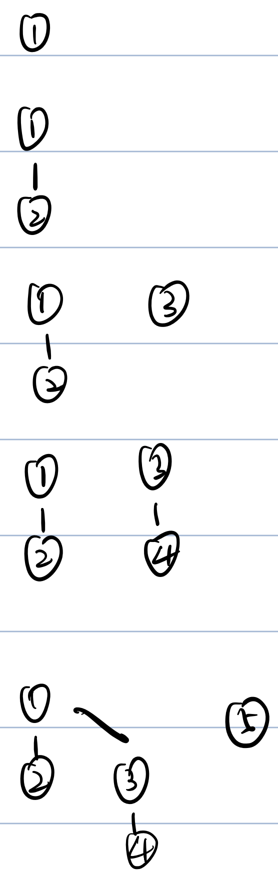
2.

3.

4.

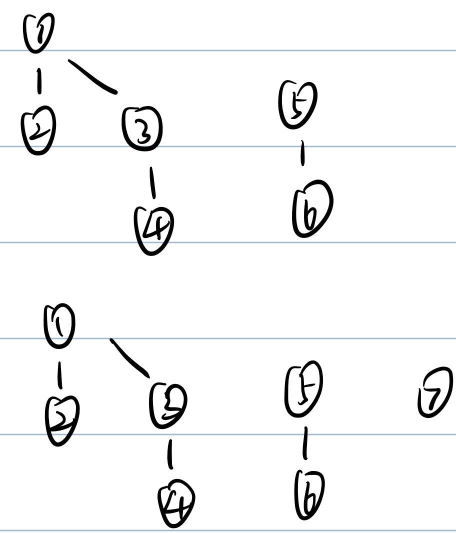
5.

(进位)



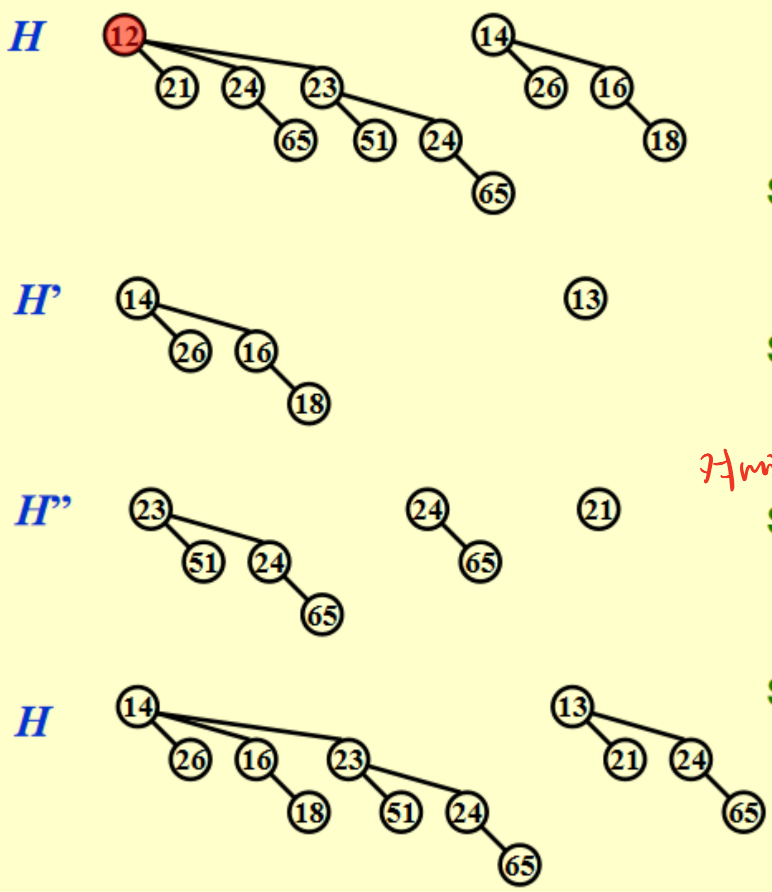
6.

7.



=> 操作N次 insert, $T_p = O(N)$ (worst), average time = $\frac{O(N)}{N}$, const

delete Min(H)



Step 1: FindMin in B_k /* O(log N) */

删除

Step 2: Remove B_k from H /* O(1) */

对min tree 去根后拆解.

Step 3: Remove root from B_k /* O(log N) */

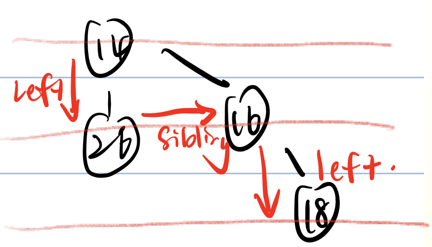
Step 4: Merge (H', H'') /* O(log N) */

进行加法.

pseudo codes.

=> 这里由于不是二叉树, 所以在构造数据结构的时候使用二叉树

时 by height.

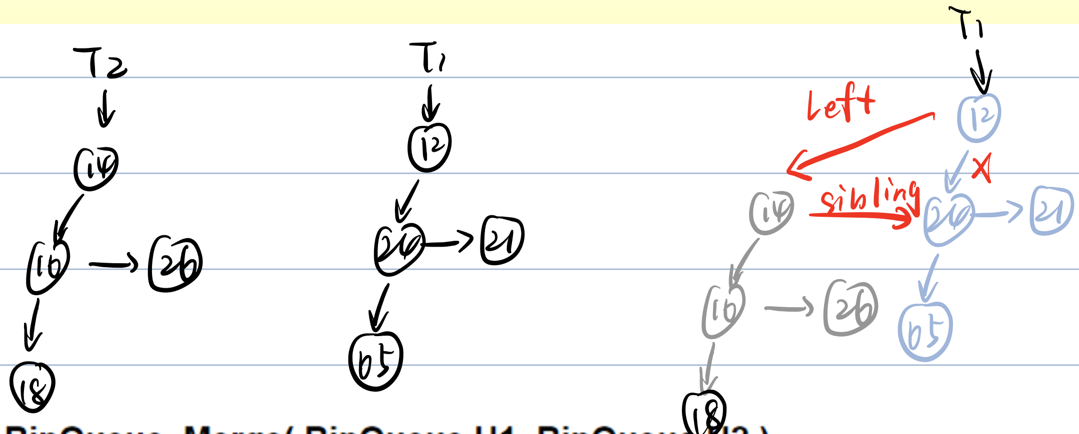



```
typedef struct BinNode *Position;
typedef struct Collection *BinQueue;
typedef struct BinNode *BinTree;  /* missing from p.176 */

struct BinNode
{
    ElementType     Element;
    Position        LeftChild;
    Position        NextSibling;
};

struct Collection
{
    int         CurrentSize;  /* total number of nodes */
    BinTree     TheTrees[ MaxTrees ];
};
```

```
BinTree
CombineTrees( BinTree T1, BinTree T2 )
{  /* merge equal-sized T1 and T2 */
    if ( T1->Element > T2->Element )
          /* attach the larger one to the smaller one */
          return CombineTrees( T2, T1 );
    /* insert T2 to the front of the children list of T1 */
    T2->NextSibling = T1->LeftChild;
    T1->LeftChild = T2;
    return T1;
}
```

大小判断，相反调换.

$$T_p = O(1)$$




```
BinQueue  Merge( BinQueue H1, BinQueue H2 )
{   BinTree T1, T2, Carry = NULL;
    int i, j;
    if ( H1->CurrentSize + H2-> CurrentSize > Capacity )  ErrorMessage();
    H1->CurrentSize += H2-> CurrentSize;
    for ( i=0, j=1; j<= H1->CurrentSize; i++, j*=2 ) {
        T1 = H1->TheTrees[i]; T2 = H2->TheTrees[i]; /*current trees */
        switch( 4*!!Carry + 2*!!T2 + !!T1 ) {   /* assign each digit to a tree */
            case 0: /* 000 */
            case 1: /* 001 */  break;
            case 2: /* 010 */  H1->TheTrees[i] = T2; H2->TheTrees[i] = NULL; break;
            case 4: /* 100 */  H1->TheTrees[i] = Carry; Carry = NULL; break;
            case 3: /* 011 */  Carry = CombineTrees( T1, T2 );
                               H1->TheTrees[i] = H2->TheTrees[i] = NULL; break;
            case 5: /* 101 */  Carry = CombineTrees( T1, Carry );
                               H1->TheTrees[i] = NULL; break;
            case 6: /* 110 */  Carry = CombineTrees( T2, Carry );
                               H2->TheTrees[i] = NULL; break;
            case 7: /* 111 */  H1->TheTrees[i] = Carry;
                               Carry = CombineTrees( T1, T2 );
                               H2->TheTrees[i] = NULL; break;
        } /* end switch */
    } /* end for-loop */
    return H1;
}
```

Carry	T2	T1

用二进制来判断

$B_0, B_1, B_2 \dots$ height

$2^0, 2^1, 2^2 \dots$ node 个数

←对i不限制，对j限制

也可以不用j值 换用log(currentsize)去限制i.

将T2表示成0或1

```
ElementType  DeleteMin( BinQueue H )
{    BinQueue DeletedQueue;
     Position DeletedTree, OldRoot;
     ElementType MinItem = Infinity;  /* the minimum item to be returned */
     int i, j, MinTree; /* MinTree is the index of the tree with the minimum item */

     if ( IsEmpty( H ) )  {  PrintErrorMessage();  return –Infinity; }
```

只在前面如果不满足直接退出

```
     for ( i = 0; i < MaxTrees; i++) {  /* Step 1: find the minimum item */
        if( H->TheTrees[i] && H->TheTrees[i]->Element < MinItem ) {
              MinItem = H->TheTrees[i]->Element;  MinTree = i;    } /* end if */
     } /* end for-i-loop */
     DeletedTree = H->TheTrees[ MinTree ];
     H->TheTrees[ MinTree ] = NULL;   /* Step 2: remove the MinTree from H => H' */
     OldRoot = DeletedTree;   /* Step 3.1: remove the root */
     DeletedTree = DeletedTree->LeftChild;   free(OldRoot);
     DeletedQueue = Initialize();   /* Step 3.2: create H" */
     DeletedQueue->CurrentSize = ( 1<<MinTree ) – 1;  /* 2^MinTree – 1 */
```

Mintree 表示层数

```
     for ( j = MinTree – 1; j >= 0; j – – ) {
```

这里mintree：除去root前完整的树有4层为3个sibling.

```
        DeletedQueue->TheTrees[j] = DeletedTree;
        DeletedTree = DeletedTree->NextSibling;
```

→ sibling 为同行横向，表示拆解

```
        DeletedQueue->TheTrees[j]->NextSibling = NULL;
     } /* end for-j-loop */
     H->CurrentSize  – = DeletedQueue->CurrentSize + 1;
     H = Merge( H, DeletedQueue ); /* Step 4: merge H' and H" */
     return MinItem;
}
```

【Claim】 A binomial queue of N elements can be built by **N successive insertions** in O(N) time.

Proof 1 (Aggregate):

$\boxed{+1}$	B_0	/*step = 1 */
$\boxed{+0}$	B_1	/*step = 1, link = 1 */
$\boxed{+1}$	B_1 B_0	/*step = 1*/
$\boxed{-1}$	B_2	/*step = 1, link = 2 */
$\boxed{+1}$	B_2 B_0	/*step = 1*/
	B_2 B_1	/*step = 1, link = 1*/
	B_2 B_1 B_0	/*step = 1*/
$\boxed{-2}$	B_3	/*step = 1, link = 3*/
	B_3 B_0	/*step = 1*/

变化的树的棵数 … … … …

Total steps = N

Total links =

$$N\left(\frac{1}{4}+2\times\frac{1}{8}+3\times\frac{1}{16}+\ldots\right)$$

$$=O(N)$$

link越少，代价越低

(∵link的过程越增加树)

step+link的值

Proof 2: An insertion that costs c units results in a net increase of $2 - c$ trees in the forest.

可以当定律

在规律中找到

变化的树的棵数.

C_i ::= cost of the ith insertion

Φ_i ::= number of trees *after* the ith insertion ($\Phi_0 = 0$)

一次插入后的树的棵树.

推出

$$C_i + (\Phi_i - \Phi_{i-1}) = 2 \quad \text{for all } i = 1, 2, \ldots, N$$

Add all these equations up $\Rightarrow$ $\sum_{i=1}^{N} C_i + \Phi_N - \Phi_0 = 2N$

定值.

$$\sum_{i=1}^{N} C_i = 2N - \Phi_N \le 2N = O(N)$$

$$T_{worst} = O(\log N), \text{ but } T_{amortized} = 2$$