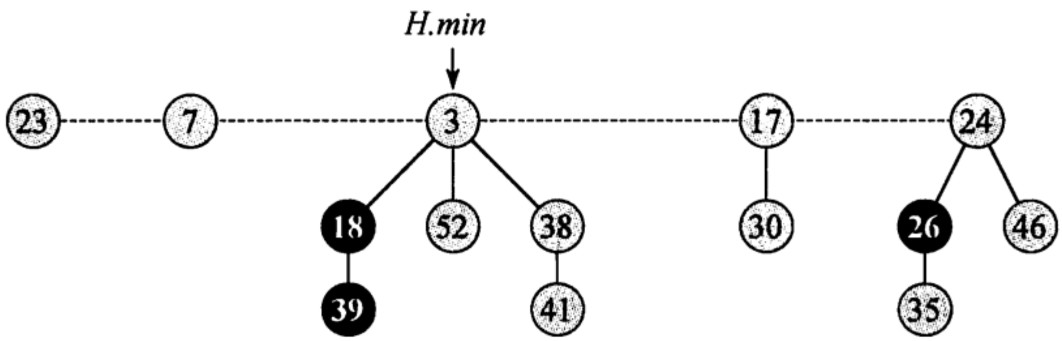
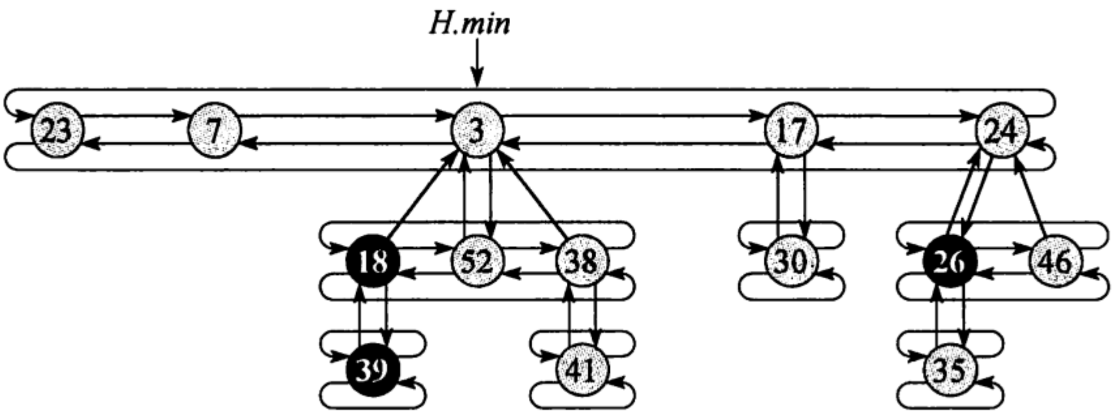
算法导论复习5——高级数据结构

Chapter 19 斐波那契堆



(a)



(b)

斐波那契堆的性能（摊还分析）

- 建堆：O(1)
- 插入：O(1)
- 取最小值：O(1)
- 提取出最小值：O(lgn)
- 合并：O(1)
- 减值：O(1)

操　作	二项堆（最坏情形）	斐波那契堆（摊还）
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$O(\lg n)$

斐波那契堆的势函数

$$\Phi(H) = t(H) + 2m(H)$$

t为森林数，m为已经标记的节点数

斐波那契堆的插入

- 将节点的degree、p、child、mark全部置为0
- 将新节点插入斐波那契堆的根链表中
- 更新min指针和节点数量

```
FIB-HEAP-INSERT(H, x)
x.degree = 0
x.p = NIL
x.child = NIL
x.mark = FALSE
if H.min == NIL
    create a root list for H containing just x
    H.min = x
else insert x into H's root list
    if x.key < H.min.key
        H.min = x
H.n = H.n + 1
```

斐波那契堆的合并

- 将两个堆的根链表合并
- 更新节点数量

```
FIB-HEAP-UNION(H1, H2)
H = MAKE-FIB-HEAP()
H.min = H1.min
concatenate the root list of H2 with the root list of H
if (H1.min == NIL) or (H2.min ≠ NIL and H2.min.key < H1.min.key)
    H.min = H2.min
H.n = H1.n + H2.n
return H
```

抽取最小节点

- 将最小节点的每个孩子都插入到根链表
- 如果抽取节点是唯一节点，将堆置为空堆，否则更新min域
- 执行CONSOLIDATE合并根链表
- 更新堆的数量

```
FIB-HEAP-EXTRACT-MIN(H)
z = H.min
if z ≠ NIL
    for each child x of z
        add x to the root list of H
        x.p = NIL
    remove z from the root list of H
    if z == z.right
        H.min = NIL
    else H.min = z.right
        CONSOLIDATE(H)
    H.n = H.n − 1
return z
```

CONSOLIDATE合并根节点的操作

- 计算最大度数的上界D(n)，其中D(n) = O(logn)
- 建立辅助数组A，A[i]指向了度数为i的根节点，初始时均为NULL
- 遍历根链表，对于每个根，通过A数组来找到和它度数相同的节点，**在保证最小堆性质下**将这些节点插入根的孩子链表中**（或者相反插入），清除掉这个节点的标记mark**，更新根的度数，并重新查找和它度数相同的节点
- 这时每个A中的位置指向了唯一的根节点，将这些根节点顺序插入堆中。

```
CONSOLIDATE(H)
let A[0..D(H.n)] be a new array
for i = 0 to D(H.n)
    A[i]=NIL
for each node w in the root list of H
    x = w
    d = x.degree
    while A[d] ≠ NIL
        y = A[d]   // another node with the same degree as x
        if x.key > y.key
            exchange x with y
        FIB-HEAP-LINK(H,y, x)
        A[d]= NIL
        d = d + 1
    A[d]= x
H.min = NIL
for i = 0 to D(H.n)
    if A[i] ≠ NIL
        if H.min == NIL
            create a root list for H containing just A[i]
            H.min = A[i]
        else insert A[i] into H's root list
            if A[i].key < H.min.key
                H.min = A[i]
```

```
FIB-HEAP-LINK(H, y, x)
remove y from the root list of H
make y a child of x, incrementing x.degree
y.mark = FALSE
```

关键字减值

- 直接对某个关键字进行减值
- 如果减值后违反了最小堆性质，就将其从剪除后插入根链表。之后使用级联切除将其父亲标记。若其父亲已经被标记，则将其父亲剪除，再继续级联切除直至根节点

```
FIB-HEAP-DECREASE-KEY(H, x, k)
if k > x.key
    error "new key is greater than current key"
x.key = k
y = x.p
if y ≠ NIL and x.key < y.key
    CUT(H, x, y)
    CASCADING-CUT(H, y)
if x.key < H.min.key
    H.min= x

CUT(H, x, y)
remove x from the child list of y, decrementing y.degree
add x to the root list of H
x.p = NIL
x.mark = FALSE

CASCADING-CUT(H,y)
z = y.p
if z ≠ NIL
    if y.mark = = FALSE
        y.mark = TRUE
    else CUT(H, y, z)
        CASCADING-CUT(H,z )
```

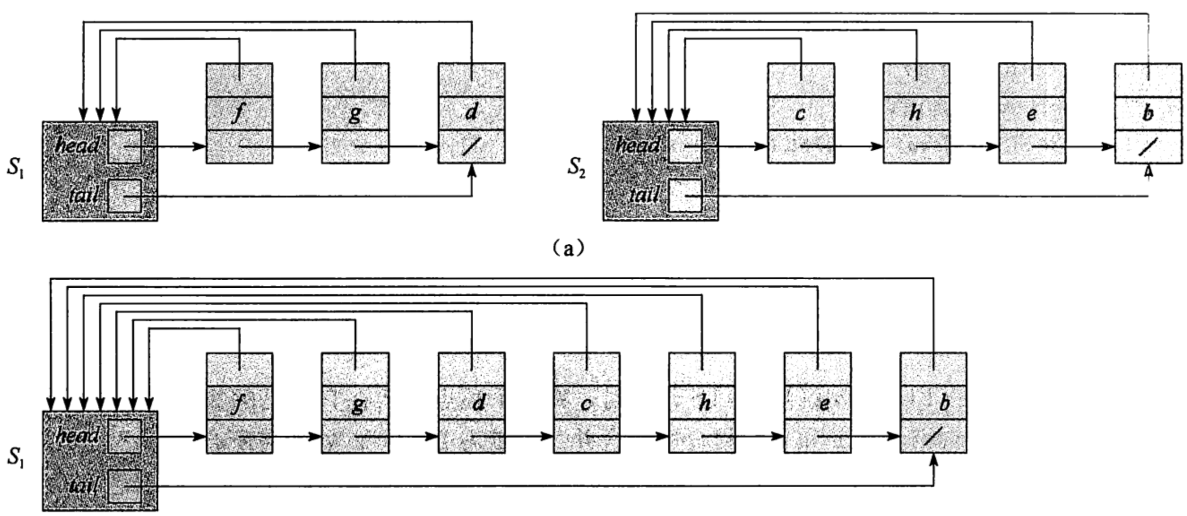
删除节点

- 将节点减值为负无穷
- 抽取最小节点

FIB-HEAP-DELETE(H, x)
1 FIB-HEAP-DECREASE-KEY(H, x, $-\infty$)
2 FIB-HEAP-EXTRACT-MIN(H)

Chapter 21 并查集

不相交集合的链表表示



(a)

(b)

- 合并的启发式策略：**总是把较短的表拼接到较长的表中**

定理 21.1 使用不相交集合的链表表示和加权合并启发式策略，一个具有 m 个 MAKE-SET、UNION 和 FIND-SET 操作的序列(其中有 n 个是 MAKE-SET 操作)需要的时间为 $O(m+n\lg n)$。

这是因为**第i次更新指针，由于是较小规模拼入较大规模，所以集合规模一定大于等于2^i**。

不相交集合森林

- 启发式策略
  - 按秩合并
  - 路径压缩

启发式策略对运行时间的影响

如果单独使用按秩合并或路径压缩，它们每一个都能改善不相交集合森林上操作的运行时间，而一起使用这两种启发式策略时，这种改善更大。单独来看，按秩合并产生的运行时间为$O(m\lg n)$（见练习 21.4-4），并且这个界是紧的（见练习 21.3-3）。尽管这里不打算来证明它，然而对于一个具有 n 个 MAKE-SET 操作（因此最多有 $n-1$ 个 UNION 操作）和 f 个 FIND-SET 操作的序列，单独使用路径压缩启发式策略给出的最坏情况运行时间为 $\Theta(n+f\cdot(1+\log_{2+f/n}n))$。

当同时使用按秩合并与路径压缩时，最坏情况的运行时间为 $O(m\alpha(n))$，这里 $\alpha(n)$ 是一个增长非常慢的函数，其定义将在 21.4 节给出。在任何一个可以想得到的不相交集合数据结构的应用中，都有 $\alpha(n)\leqslant 4$；因此，我们可以认为在所有实际应用中，其运行时间与 m 呈线性关系。然而，严格地说，它是超线性的。21.4 节将证明这个上界。

- 实现代码
  - 按秩合并

```
UNION(x, y)
1   LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
1   if x.rank > y.rank
2       y.p = x
3   else x.p = y
4       if x.rank == y.rank
5           y.rank = y.rank + 1
```

  - 路径压缩

```
FIND-SET(x)
1   if x ≠ x.p
2       x.p = FIND-SET(x.p)
3   return x.p
```