Algorithm Theory

08 – Fibonacci Heaps

Dr. Alexander Souza

Winter term 11/12

Priority queues: operations

Priority queue *Q*

Operations:

Q.initialize(): initializes an empty queue *Q*

Q.isEmpty(): returns true iff *Q* is empty

Q.insert(e): inserts element *e* into *Q* and returns a pointer to the node containing *e*

Q.deletemin(): returns the element of *Q* with minimum key and deletes it

Q.min(): returns the element of Q with minimum key

Q.decreasekey(v,k): decreases the value of *v*'s key to the new value *k*

Priority queues: operations

Additional operations:

Q.delete(v) : deletes node *v* and its element from *Q*
(without searching for *v*)

Q.meld(Q´): unites *Q* and *Q´* (concatenable queue)

Q.search(k) : searches for the element with key *k* in *Q*
(searchable queue)

And many more, e.g. *predecessor, successor, max, deletemax*

Priority queues: implementations

	List	Heap	Bin. – Q.	Fib.-Hp.
insert	O(1)	O(log n)	O(log n)	O(1)
min	O(n)	O(1)	O(log n)	O(1)
delete-min	O(n)	O(log n)	O(log n)	O(log n)*
meld (m≤n)	O(1)	O(n) or O(m log n)	O(log n)	O(1)
decr.-key	O(1)	O(log n)	O(log n)	O(1)*

*= amortized cost

Q.delete(e) = Q.decreasekey(e, -∞) + Q.deletemin( )

Fibonacci heaps

„Lazy-meld“ version of binomial queues:

The melding of trees having the same order is delayed until the next deletemin operation.

Definition

A Fibonacci heap *Q* is a collection heap-ordered trees.

Variables

Q.min: root of the tree containing the minimum key

Q.rootlist: circular, doubly linked, unordered list containing the roots of all trees

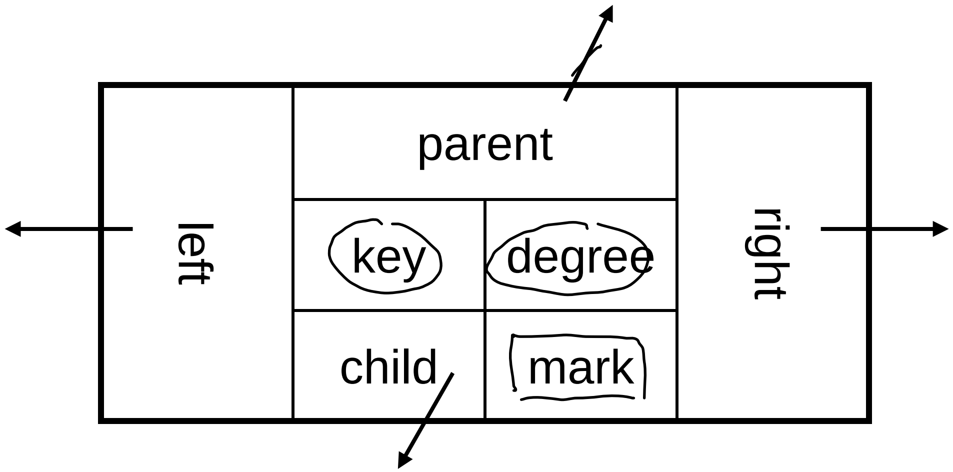
Q.size: number of nodes currently in *Q*

Trees in Fibonacci heaps

Let *B* be a heap-ordered tree in *Q.rootlist*:

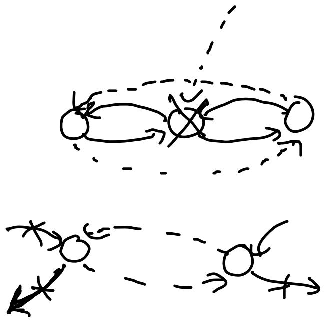
B.childlist: circular, doubly linked and unordered list of the children of B

Structure of a node



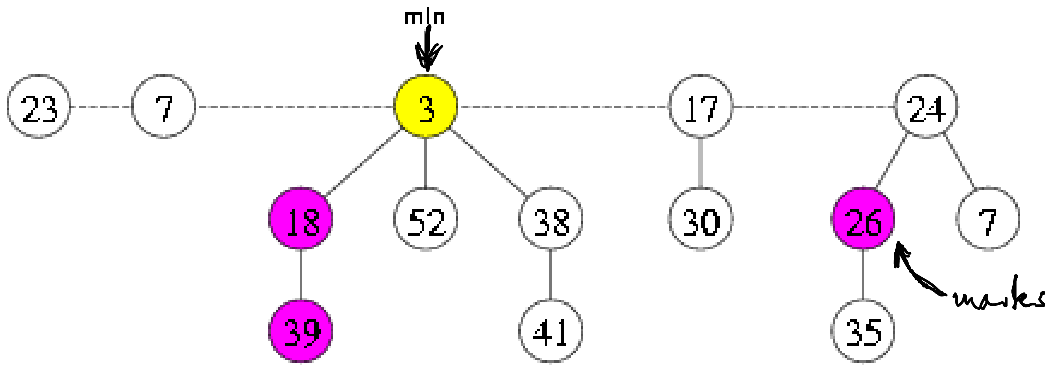
Advantages of circular, doubly linked lists:

1. Deleting a~~n~~ given element takes constant time.
2. Concatenating two lists takes constant time.



Implementation of Fibonacci heaps: Example

size

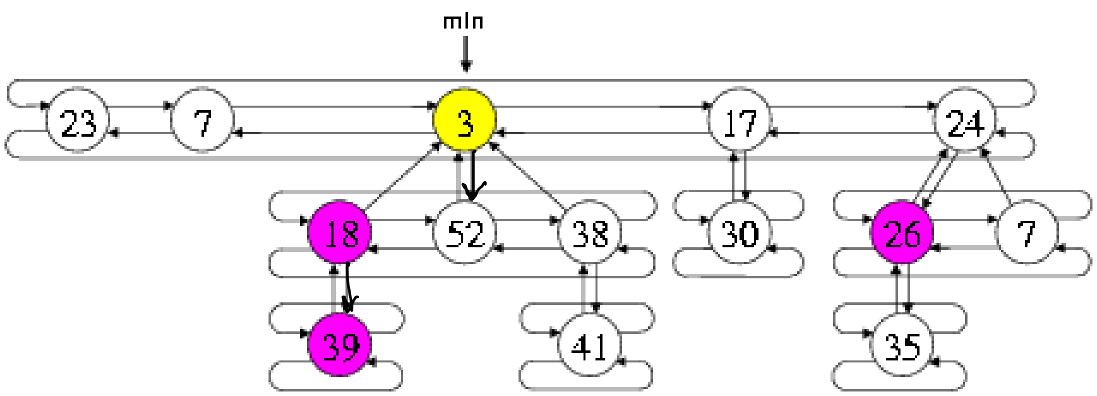


marks

heap property is maintained

Mark: Indicates that node has lost a child since the last time that node was made child of some other node.

rootlist



Operations on Fibonacci heaps

Q.initialize(): *Q.rootlist = Q.min = null*   Q.size = 0

$O(1)$

Q.meld(Q´):

1. concatenate *Q.rootlist* and *Q´.rootlist*
2. update *Q.min*

$O(1)$

Q.insert(e):

1. generate a new node with element $e \rightarrow Q'$
2. *Q.meld(Q´)*

$O(1)$

Q.min():

return *Q.min.key*

$O(1)$

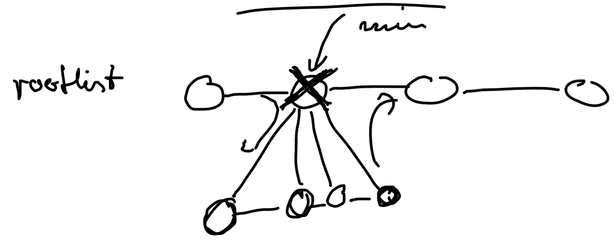
all operations here have constant running time.

Fibonacci heaps: ‘deletemin’

Q.deletemin()

/*Delete the node with minimum key from *Q* and return its element.*/

```
m = Q.min()
if Q.size() > 0
     then remove Q.min() from Q.rootlist
          add Q.min.childlist to Q.rootlist
     Q.consolidate()
   /* Repeatedly meld nodes in the root list having the same
      degree. Then determine the element with minimum key. */
return m
```



Fibonacci heaps: maximum degree of a node

rank(v) = degree of node *v* in *Q*

rank(Q) = maximum degree of any node in *Q*

Assumption: size = n

$$rank(Q) \le 2 \cdot \log n,$$

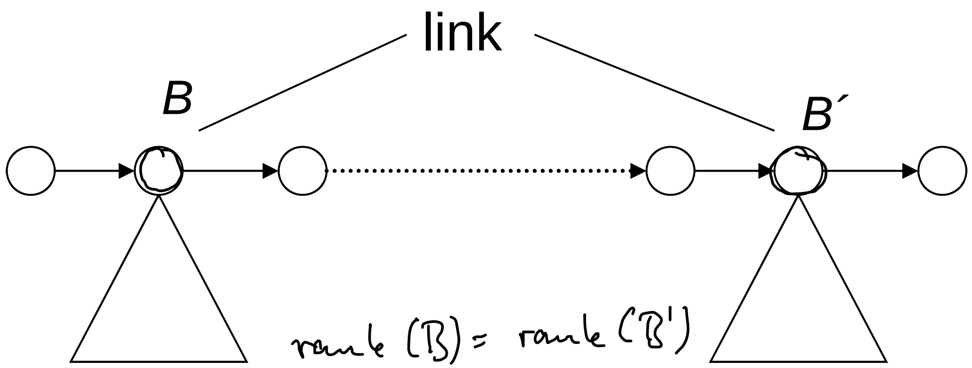
if *Q.size* = *n.*

We take this for granted at the moment.

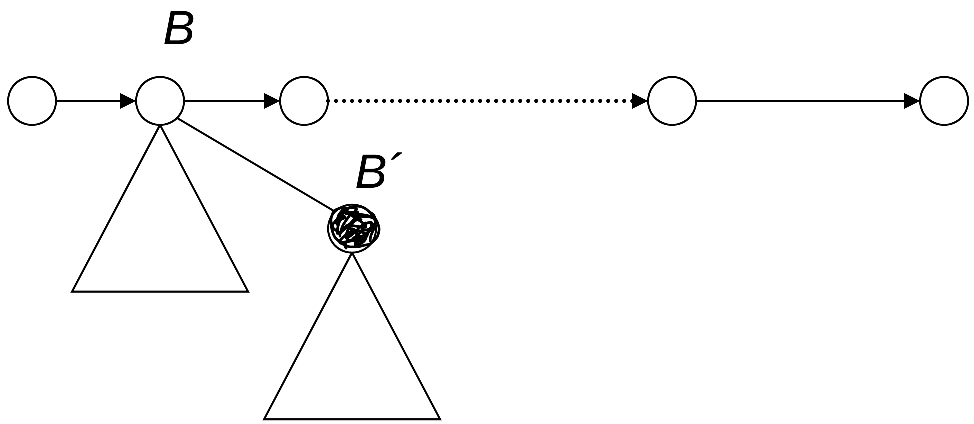
Fibonacci heaps: operation ‘link’

rank(B) = degree of the root of *B*
Heap-ordered trees *B,B´* with *rank(B)* = *rank(B´)*

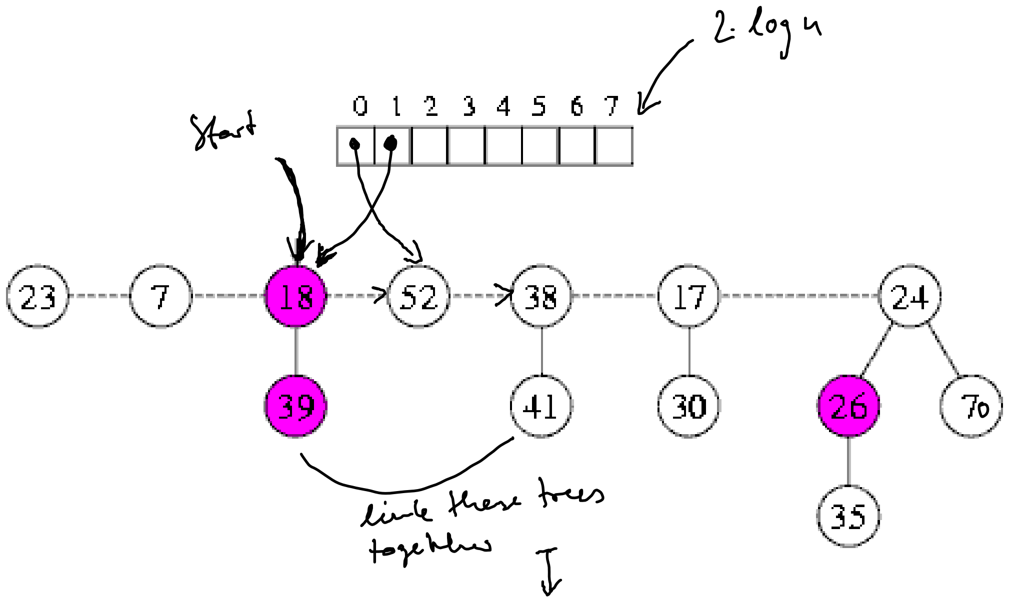
Mark = true if and only if node has lost a child since the last time the node became child of some other node.

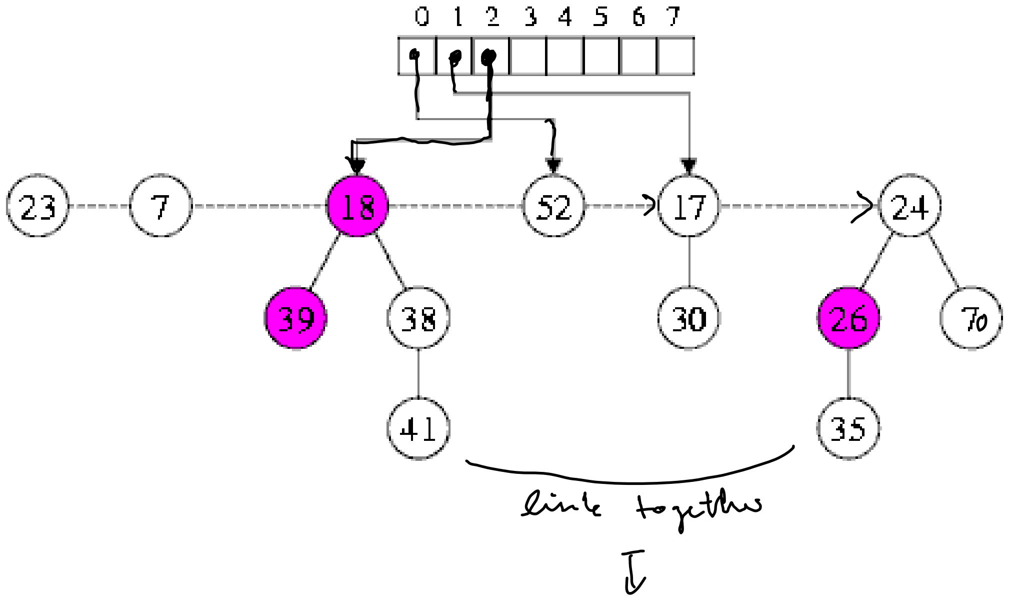


1. *rank(B)* = *rank(B)* + 1
2. *B´.mark* = *false*

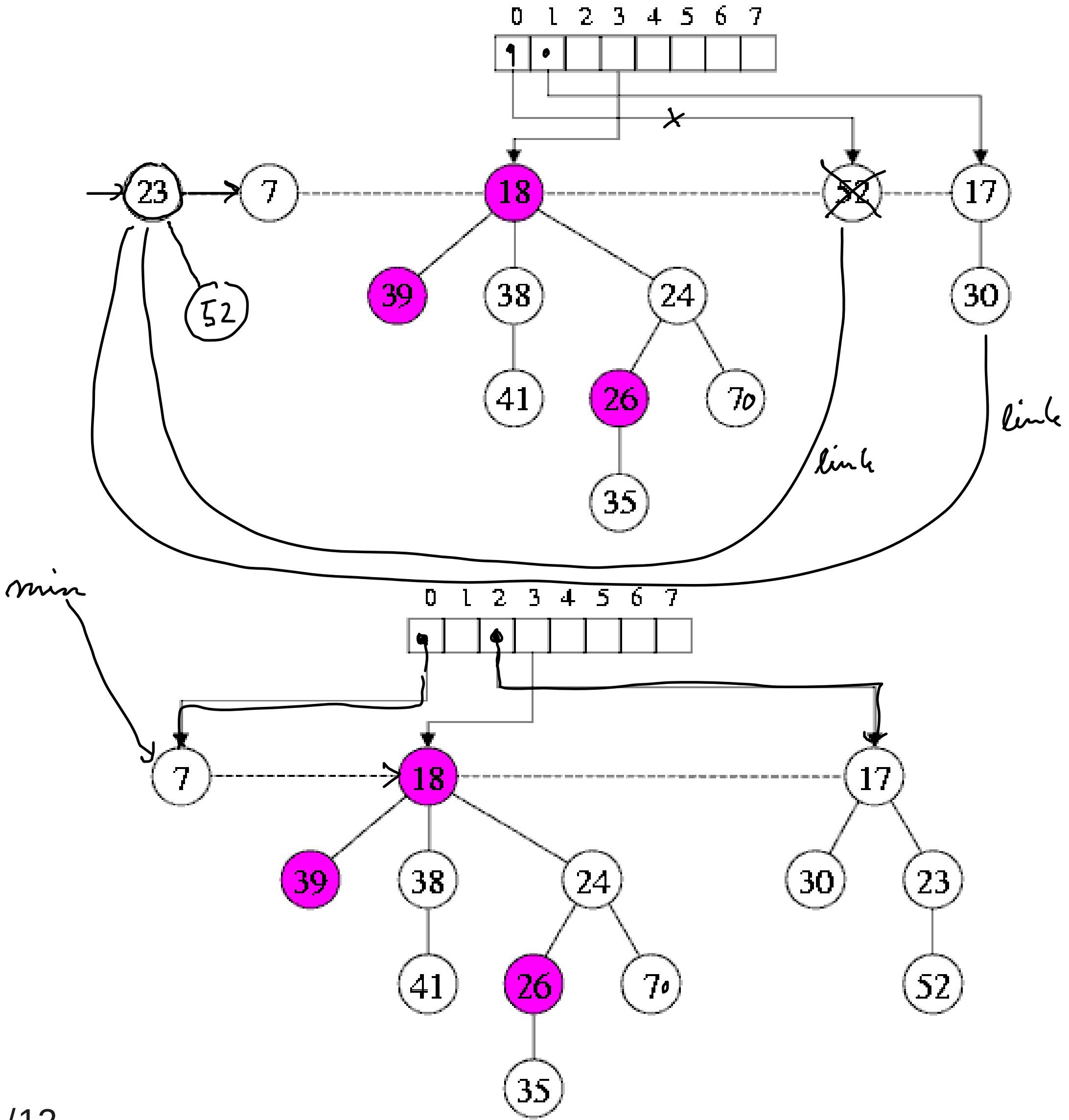


Consolidation of the root list



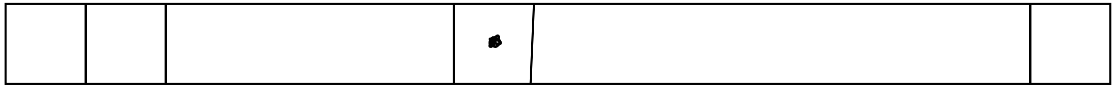


Consolidation of the root list



Fibonacci heaps: ‘deletemin’

Find roots having the same rank:
Array *A*:



0 1 2 log *n*

Q.consolidate()

```
A = array of length 2 log n pointing to Fibonacci heap nodes
for i = 0 to 2 log n do A[i] = null
while Q.rootlist ≠ ∅ do
        B = Q.delete-first()
        while A[rank(B)] is not null do
                B´ = A[rank(B)];   A[rank(B)] = null;   B = link(B,B´)
        end while
        A[rank(B)] = B
end while
determine Q.min
```