Priority Queues(heaps)

> delete the element with the highest \ lowest priority

ADT Model

Objects: A finite ordered list with zero or more elements.

Operations:

- PriorityQueue Initialize(int MaxElements);
- void Insert(ElementType X, PriorityQueue H);
- ElementType DeleteMin(PriorityQueue H);
- ElementType FindMin(PriorityQueue H);

Simple Implementation

- Array :
  - Insertion — add one item at the end ~ $\Theta(1)$
  - Deletion — find the largest \ smallest key ~ $\Theta(n)$
    remove the item and shift array ~ $O(n)$
- Linked List :
  - Insertion — add to the front of the chain ~ $\Theta(1)$
  - Deletion — find the largest \ smallest key ~ $\Theta(n)$
    remove the item ~ $\Theta(1)$

- Ordered Array :
  - Insertion — find the proper position ~ $O(n)$
    shift array and add the item ~ $O(n)$
  - Deletion — remove the first \ last item ~ $\Theta(1)$
- Ordered Linked List :
  - Insertion — find the proper position ~ $O(n)$
    add the item ~ $\Theta(1)$
  - Deletion — remove the first \ last item ~ $\Theta(1)$

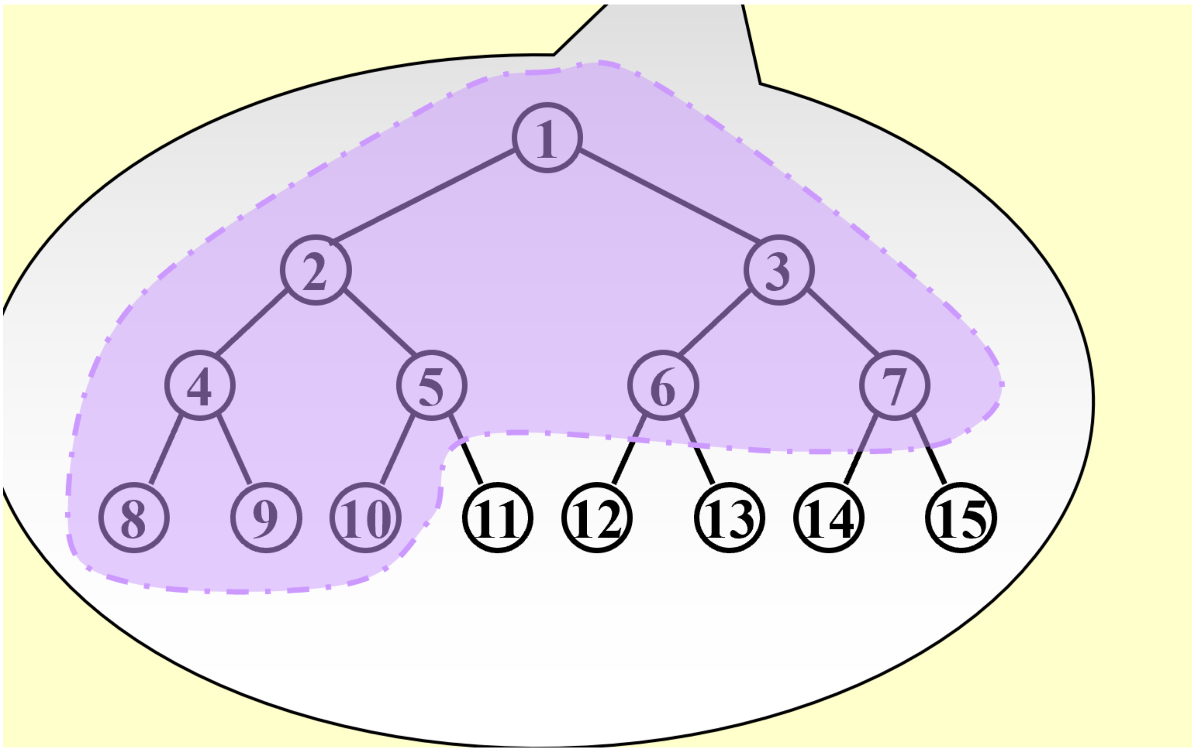
> Linked list较优，因为Insertion总是比Deletion多

Binary Heap

Structure Property

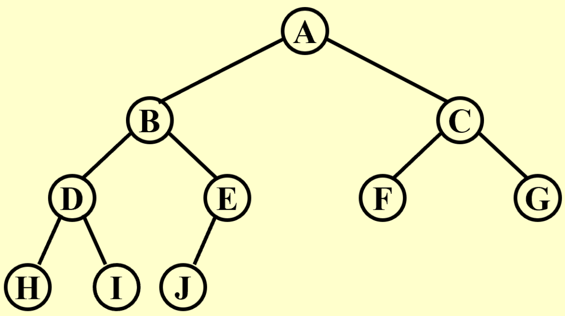
【Definition】A binary tree with n nodes and height h is complete iff its nodes correspond to the nodes numbered from 1 to n in the perfect binary tree of height h.

A complete binary tree of height h has between 2^h and $2^{h+1}-1$ nodes. $\longrightarrow$ $h = \lfloor \log N \rfloor$



- levelorder连续

❖ Array Representation : BT [n + 1] (BT [0] is not used)



BT

0	1	2	3	4	5	6
	A	B	C	D	E	F

7	8	9	10	11	12	13
G	H	I	J			

【Lemma】If a complete binary tree with n nodes is represented sequentially, then for any node with index i, $1 \le i \le n$, we have:

$$(1)\ \text{index of } parent(i) = \begin{cases} \lfloor i/2 \rfloor & \text{if } i \ne 1 \\ \text{None} & \text{if } i = 1 \end{cases}$$

$$(2)\ \text{index of } left\_child(i) = \begin{cases} 2i & \text{if } 2i \le n \\ \text{None} & \text{if } 2i > n \end{cases}$$

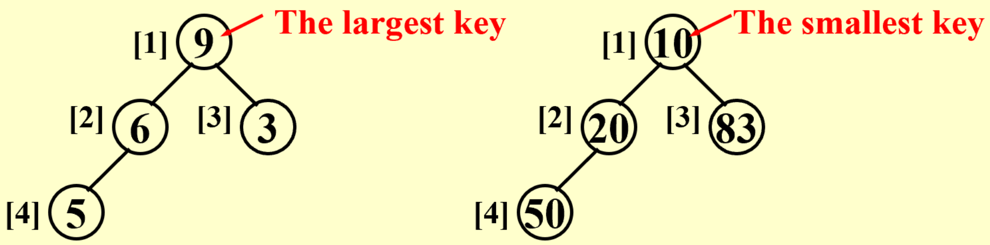
$$(3)\ \text{index of } right\_child(i) = \begin{cases} 2i+1 & \text{if } 2i+1 \le n \\ \text{None} & \text{if } 2i+1 > n \end{cases}$$

```c
PriorityQueue  Initialize( int  MaxElements )
{
    PriorityQueue  H;
    if ( MaxElements < MinPQSize )
            return  Error( "Priority queue size is too small" );
    H = malloc( sizeof ( struct HeapStruct ) );
    if ( H ==NULL )
            return  FatalError( "Out of space!!!" );
    /* Allocate the array plus one extra for sentinel */
    H->Elements = malloc(( MaxElements + 1 ) * sizeof( ElementType ));
    if ( H->Elements == NULL )
            return  FatalError( "Out of space!!!" );
    H->Capacity = MaxElements;
    H->Size = 0;
    H->Elements[ 0 ] = MinData;  /* set the sentinel */
    return  H;
}
```

Heap Order Property

【Definition】A **min tree** is a tree in which the key value in each node is no larger than the key values in its children (if any). A **min heap** is a **complete** binary tree that is also a min tree.

Note: Analogously, we can declare a *max* heap by changing the heap order property.



A max heap　　A min heap

Operations

- Insertion
    1. 先插入到正确位置，满足Structure Property
    2. 然后通过不断与parent比较、交换，满足Heap Order Property
    - 加速swap：直接overwrite

```
/* H->Element[ 0 ] is a sentinel */
void  Insert( ElementType  X,  PriorityQueue  H )
{
    int  i;

    if ( IsFull( H ) ) {
        Error( "Priority queue is full" );
        return;
    }

    for ( i = ++H->Size; H->Elements[ i / 2 ] > X; i /= 2 )
        H->Elements[ i ] = H->Elements[ i / 2 ];

    H->Elements[ i ] = X;
}
```

H->Element[0] is a *sentinel* that is no larger than the minimum element in the heap.

Faster than *swap*

$T(N) = O(\log N)$

> 由于0放了最小值，因此一定会在根节点停下

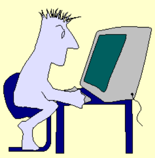
- DeleteMin

1. 先把最末尾的元素换到根节点，确保Structure Property
2. 然后向下比较、交换，确保Heap Order Property
    - 子节点之间先比较，确定向下的路径
    - 交换
    - 继续向下重复过程

```c
ElementType  DeleteMin( PriorityQueue  H )
{
    int  i, Child;
    ElementType  MinElement, LastElement;
    if ( IsEmpty( H ) ) {
        Error( "Priority queue is empty" );
        return  H->Elements[ 0 ];   }
    MinElement = H->Elements[ 1 ];  /* save the min element */
    LastElement = H->Elements[ H->Size-- ];  /* take last and reset size */
    for ( i = 1; i * 2 <= H->Size; i = Child ) {  /* Find smaller child */
        Child = i * 2;
        if (Child != H->Size && H->Elements[Child+1] < H->Elements[Child])
               Child++;
        if ( LastElement > H->Elements[ Child ] )   /* Percolate one level */
               H->Elements[ i ] = H->Elements[ Child ];
        else     break;   /* find the proper position */
    }
    H->Elements[ i ] = LastElement;
    return  MinElement;
}
```

Other Operations

☞ DecreaseKey (P, Δ, H)   *Percolate up*



sys. admin.

Lower the value of the key in the heap H at position P by a positive amount of Δ……so my programs can run with highest priority ☺.

☞ IncreaseKey (P, Δ, H)   *Percolate down*

sys. admin.

Increases the value of the key in the heap H at position P by a positive amount of Δ……drop the priority of a process that is consuming excessive CPU time.

☞ Delete (P, H)　DecreaseKey(P, ∞, H); DeleteMin(H)

sys. admin.

Remove the node at position P from the heap H …… delete the process that is terminated (abnormally) by a user.

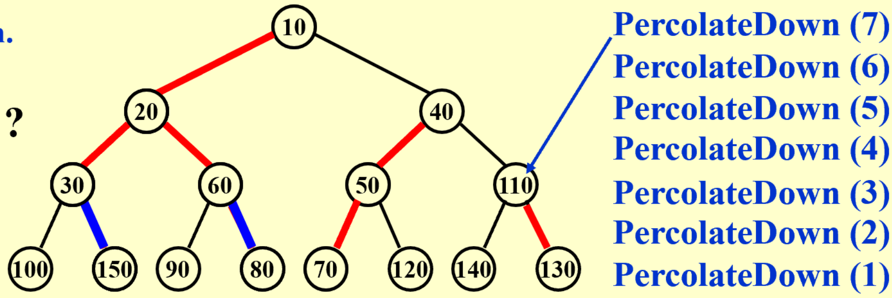
☞ BuildHeap (H)　N successive Insertions ?

sys. admin.

Place N input keys into an empty heap H.

150, 80, 40, 30, 10, 70, 110, 100, 20, 90, 60, 50, 120, 140, 130

$T(N) = ?$



> Insertions太慢了
>
> 先build tree 满足Structure Property
>
> 再从倒数第二层开始做PercolateDown

【Theorem】For the perfect binary tree of height h containing $2^{h+1} - 1$ nodes, the sum of the heights of the nodes is $2^{h+1} - 1 - (h + 1)$.

$$\longrightarrow T(N) = O(N)$$

Application

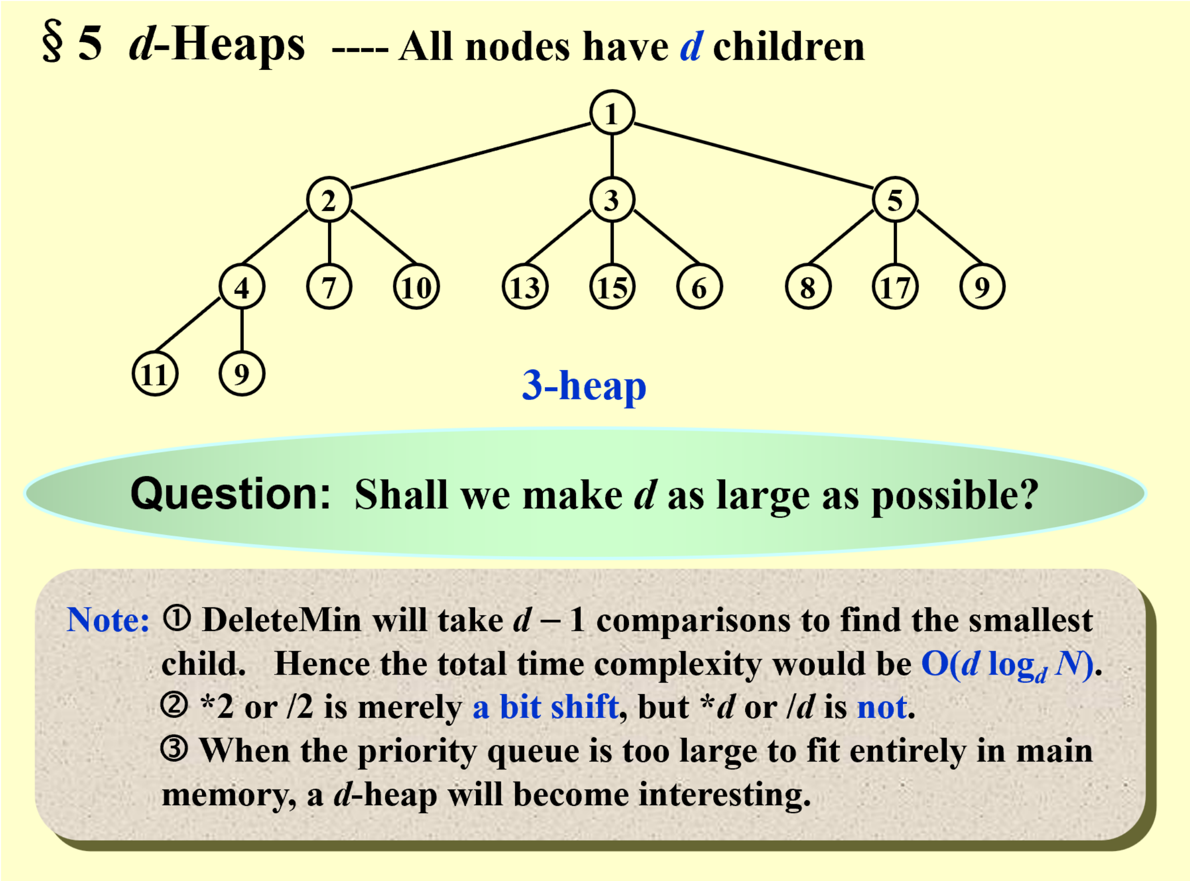
〖Example〗 Given a list of N elements and an integer k. Find the kth largest element.

> 建堆+k次delete

- 与Qsort的选择

  全排序用Qsort，部分排序（前1000项）用heapSort

d-Heaps



- pros: 高度减小
- cons: delete的比较次数会增多

2-3 分数 2

If a d-heap is stored as an array, for an entry located in position i, the parent, the first child and the last child are at:

- A. $\lceil (i+d-2)/d \rceil$, $(i-2)d+2$, and $(i-1)d+1$
- B. $\lceil (i+d-1)/d \rceil$, $(i-2)d+1$, and $(i-1)d$
- C. $\lfloor (i+d-2)/d \rfloor$, $(i-1)d+2$, and $id+1$
- D. $\lfloor (i+d-1)/d \rfloor$, $(i-1)d+1$, and id

答案正确：2 分

创建提问