计算机问题求解 – 论题2-10

- 堆与堆排序

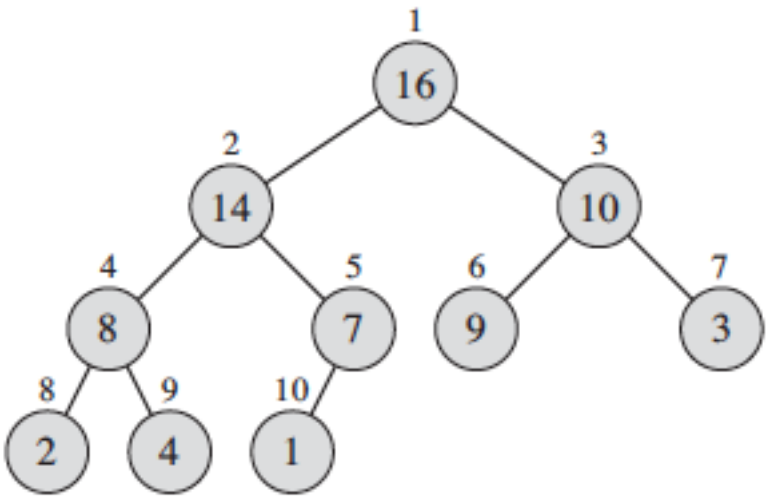
课程研讨

- TC第6章

问题1： heap和heapsort

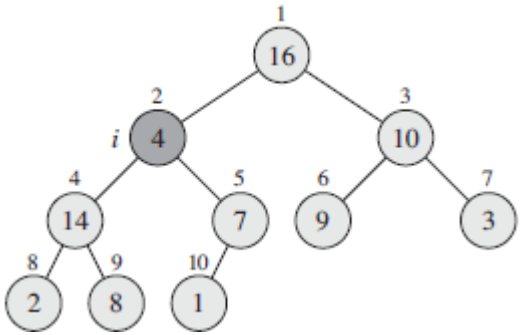
- 什么是堆？
- 它擅长于dynamic set的哪些操作？

Search
Insert
Delete
Minimum
Maximum
Successor
Predecessor



问题1： heap和heapsort (续)

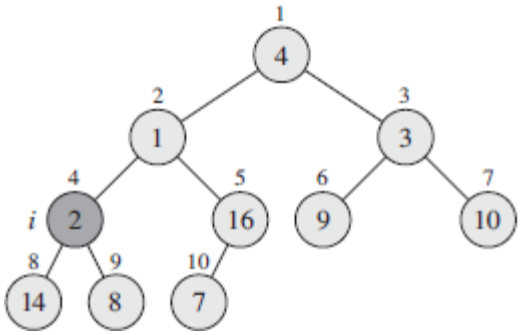
```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```



- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 你能解释它的运行时间吗？ $T(n) \le T(2n/3) + \Theta(1)$

问题1： heap和heapsort (续)

```
BUILD-MAX-HEAP(A)
A.heap-size = A.length
for i = ⌊A.length/2⌋ downto 1
    MAX-HEAPIFY(A, i)
```

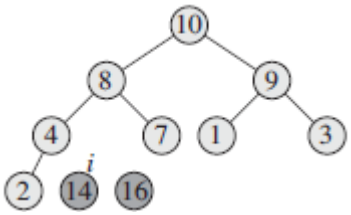


- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 你能解释它的运行时间吗？

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right)$$

问题1： heap和heapsort (续)

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i = A.length downto 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size − 1
    MAX-HEAPIFY(A, 1)
```



- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 你能解释它的运行时间吗？$O(n \lg n)$

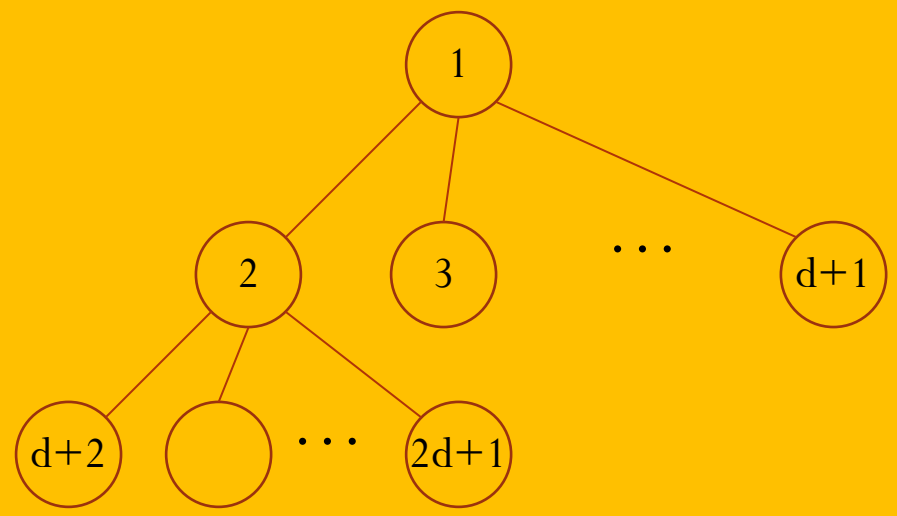
- 假设A中元素各不相同，你能否构造一种初始序，使得运行时间少于nlgn？

d-ary Heap

- A *d-ary heap* is like a binary heap, but non-leaf nodes have d children instead of 2 (with one possible exception)
  1. How to represent a d-ary heap in an array?
  2. Heap height = ? (in terms of n and d)
  3. Implement EXTRACT-MAX
  4. Implement INSERT
  5. Implement INCREASE-KEY(A,I,k), running time? (in terms of d and n)

Q1. Represent d-ary in an Array

- Binary Heap
  - Root (i) = 1
  - PARENT(i) =
  - LEFT(i) = 2i,
- d-ary heap
  - PARENT(i) =
  - j^{th} child(i) = d



$$\text{d-ary-PARENT}(i) = \lfloor (i-2)/d + 1 \rfloor$$

$$\text{d-ary-CHILD}(i, j) = d(i-1) + j + 1$$

Q2. Height of d-ary heap

- By intuition: $\Theta(\log_d n)$
- Calculating the height

$$
\begin{aligned}
1 + d + d^2 + \ldots + d^{h-1} &< n \leq 1 + d + d^2 + \ldots + d^h \\
\frac{d^h - 1}{d - 1} &< n \leq \frac{d^{h+1} - 1}{d - 1} \\
d^h &< n(d-1) + 1 \leq d^{h+1} \\
h &< \log_d(n(d-1) + 1) \leq h + 1
\end{aligned}
$$

Q3. EXTRACT-MAX

- Directly get max by A[1]
  - By property of Heap
- Maintain the Heap property
  - By HEAPIFY(A,i,n,d) (fixHeap)

```
EXTRACT-MAX(A, n)
max ← A[1]
A[1] ← A[n]
n = n − 1
HEAPIFY(A, 1, n, d)
return max
```

HEAPIFY

- After EXTRACT-MAX the heap property is only partially ruined
- Fix the partially ruined sub-tree recursively

```
HEAPIFY(A, i, n, d)
    j->i
    For (k = each child of A[i])
        if (Index[Child] <= n and Child > A[j])
        then j=Index[Child]
    If (j!=i)
    then
        Exchange A[i] <-> A[j]
        HEAPIFY(A, j, n, d)
```

$\Theta\ (d\log_d n)$

Q4. INSERT

- Directly append the element to the Heap
- The Heap property is partially ruined
- Fix the partially ruined heap

```
INSERT(A, k, n, d)
n ← n + 1
A[n] = −∞
INCREASE-KEY(A, i, k, n)
```

Q5. INCREASE-KEY

- Partially ruined heap property
- The increased key goes up like a bubble

```
INCREASE-KEY(A, i, k)
A[i] <- max(A[i], k)
If (k ==A[i])
    while (i>1 and PARENT(i) < A[i])
        do
            Exchange A[i] <-> PARENT(i)
            i <- PARENT(i)
```

$\Theta(\log_d n)$

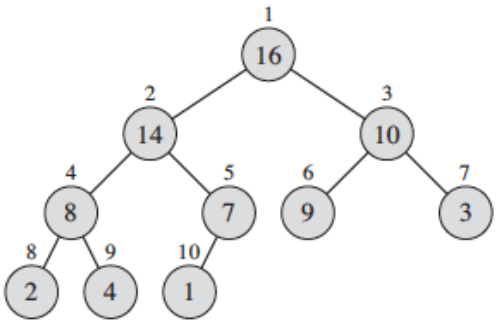
问题2： priority queue

- 什么是优先队列？
- 它擅长于dynamic set的哪些操作？

Search
Insert
Delete
Minimum
Maximum
Successor
Predecessor

问题2： priority queue (续)

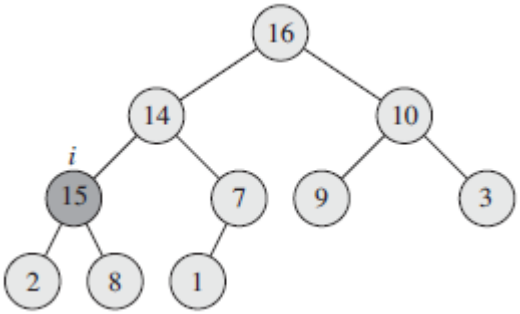
```
HEAP-EXTRACT-MAX(A)
if A.heap-size < 1
    error "heap underflow"
max = A[1]
A[1] = A[A.heap-size]
A.heap-size = A.heap-size - 1
MAX-HEAPIFY(A, 1)
return max
```



- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 你能解释它的运行时间吗？ $O(\lg n)$

问题2： priority queue (续)

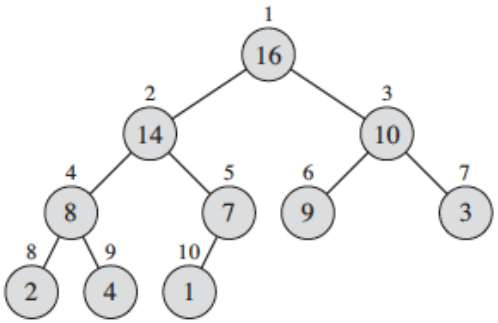
```
HEAP-INCREASE-KEY(A, i, key)
if key < A[i]
    error "new key is smaller than current key"
A[i] = key
while i > 1 and A[PARENT(i)] < A[i]
    exchange A[i] with A[PARENT(i)]
    i = PARENT(i)
```



- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 你能解释它的运行时间吗？ $O(\lg n)$

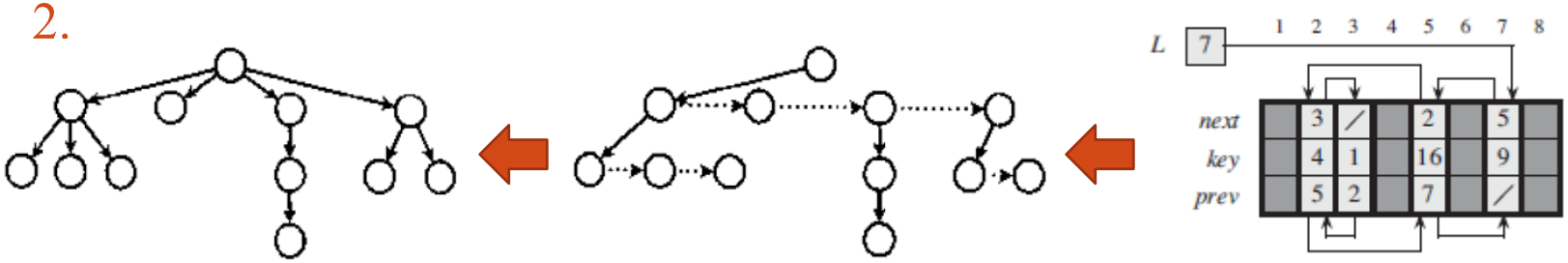
问题2： priority queue (续)

```
MAX-HEAP-INSERT(A, key)
A.heap-size = A.heap-size + 1
A[A.heap-size] = −∞
HEAP-INCREASE-KEY(A, A.heap-size, key)
```



- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 你能解释它的运行时间吗？ $O(\lg n)$

问题3： ADT

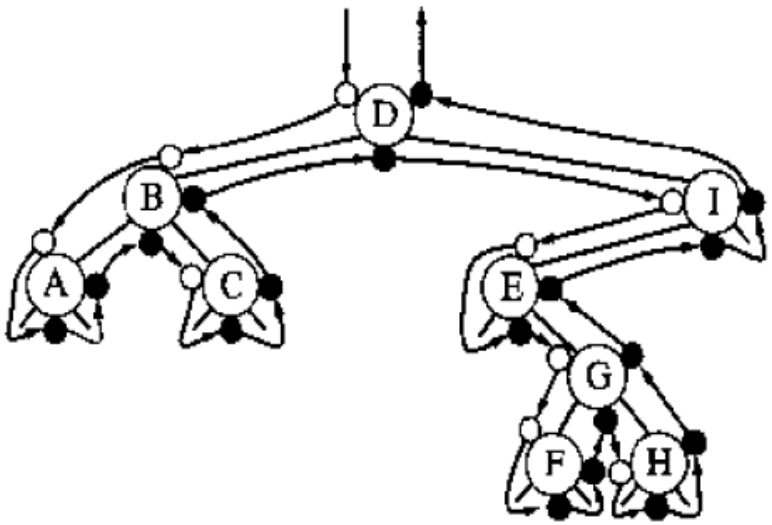
1. priority queue ← heap ← array
2. 

结合这两个例子，谈谈你对ADT的理解

- ADT和分层抽象对于算法的设计与分析有什么好处？
  - 设计：信息隐藏/数据封装、性能优化
  - 分析：正确性分析、性能分析

问题3： ADT (续)

```
void  traverse(BinTree T)
    if (T is not empty)
        Preorder-process root(T);
        traverse(leftSubtree(T));
        Inorder-process root(T);
        traverse(rightSubtree(T));
        Postorder-process root(T);
    return;
```



- 你理解binary tree的preorder/inorder/postorder了吗？
- 它们遍历的顺序分别是什么？

问题3： ADT (续)

- 你理解union-find (disjoint sets)了吗？

UnionFind create(int n)

Precondition: none.

Postconditions: If sets = create(n), then sets refers to a newly created object; find(sets, e) = e for $1 \leq e \leq n$, and is undefined for other values of e.

int find(UnionFind sets, e)

Precondition: Set $\{e\}$ has been created in the past, either by makeSet(sets, e) or create.

void makeSet(UnionFind sets, int e)

Precondition: find(sets, e) is undefined.

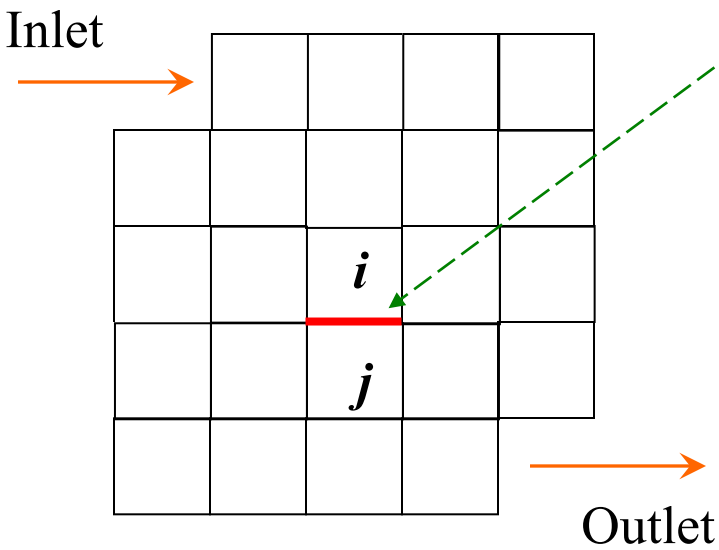
Postconditions: find(sets, e) = e; that is, e is the set id of a singleton set containing e.

void union(UnionFind sets, int s, int t)

Preconditions: find(sets, s) = s and find(sets, t) = t, that is, both s and t are set ids, or "leaders." Also, $s \neq t$.

Postconditions: Let /sets/ refer to the state of sets before the operation. Then for all x such that find(/sets/, x) = s, or find(/sets/, x) = t, we now have find(sets, x) = u. The value of u will be either s or t. All other find calls return the same value as before the union operation.

Maze Creating: an Example



Selecting a wall to pull down randomly

If *i,j* are in same equivalence class, then select another wall to pull down, otherwise, joint the two classes into one.

The maze is complete when the inlet and outlet are in one equivalence class.

Counting Inversions

- Given an array of n (distinct) numbers, how many inversions?
  - E.g.: 1,2,7,5, 6. (7,5), (7,6) are inversions

- Simple solution
  - $O(n^2)$

Divide and Conquer

- Simple division inspired by merge sort
- Counting Inversions in $O(n \lg n)$

Algorithm 2 Sort-and-Count(L)

```
if L contains one element then
    return zero;
end if
Divide L into two halves: A contains the first ⌈n/2⌉ elements, B contains
the remaining ⌊n/2⌋;
(r_A, A) = Sort-and-Count(A);
(r_B, B) = Sort-and-Count(B);
(r, L) = Merge-and-Count(A,B);
return r = r_A + r_B + r, and the sorted list L;
```

Question: Anagram Problem

- An anagram is a word that is made by changing the order of the letters in another word. For example, ate, tea and eat belong to one set of anagrams. Design an algorithm for finding all the set of anagrams in a large file of English words.
- Time complexity $O(n \lg n)$

1. Sort all the words according to their length. $O(n)$
2. Mark all the words with ID (the sorted "word"). $O(n)$
3. Sorting words according to ID, and merge the same words. $O(n \log n)$

k Largest Numbers

- Given a set of n numbers, we wish to find the k largest numbers in sorted order, using a comparison-based algorithm.
- Sorting and getting the last k numbers.
  - $O(n \log n)$
- Using a heap and exacting the first k numbers.
  - $O(n + k \log n)$
- Getting the kth largest number, then partitioning and sorting.
  - $O(n + k \log k)$

Finding Two Elements

- Both A and B are two arrays with n integers. For a given integer x, design an algorithm to find out whether exists $a \in A$ and $b \in B$, satisfying $x = a + b$. Analyze the complexity of your solution.

扔鸡蛋问题

- 假设有n级台阶，现在从台阶上往下扔鸡蛋，在某层台阶之上往下摔，鸡蛋就会摔破。问至少要测试多少次才能测出这个层数？
  - 如果只有一个鸡蛋
  - 如果有无穷多个相同鸡蛋
  - 如果有两个相同鸡蛋