计算机问题求解 – 论题2–9

- 堆与堆排序

课程研讨

- TC第6章
- SB第2章

分治法（Divide and Conquer）

solve(I)
    n=size(I);
    if ($n \le$ smallSize)
        solution=directlySolve(I)
    else
        divide I into $I_1, \ldots I_k$;
        for each $i \in \{1, \ldots, k\}$
            S_i=solve(I_i);
        solution=combine($S_1, \ldots, S_k$);
    return solution

$T(n)=B(n)$ for $n \le$ smallSize

$$T(n)=D(n)+\sum_{i=1}^{k} T(size(I_i))+C(n)$$
for $n >$ smallSize

最小未出现自然数

- n个大小各不相同的自然数，找出不在这个自然数序列中出现的最小自然数。
  - “1、2、4、5”中最小未出现是3。

- 分别就下面两种情况设计算法
  - 若n个元素是已排序的
  - 若n个元素是未排序的

Finding the “Heavy” Element

- Find the element *i* with *freq(i)* > *n*/2 in an array of *n* elements. Here, *freq(i)* is defined as the number of occurrence of *i* in the array.

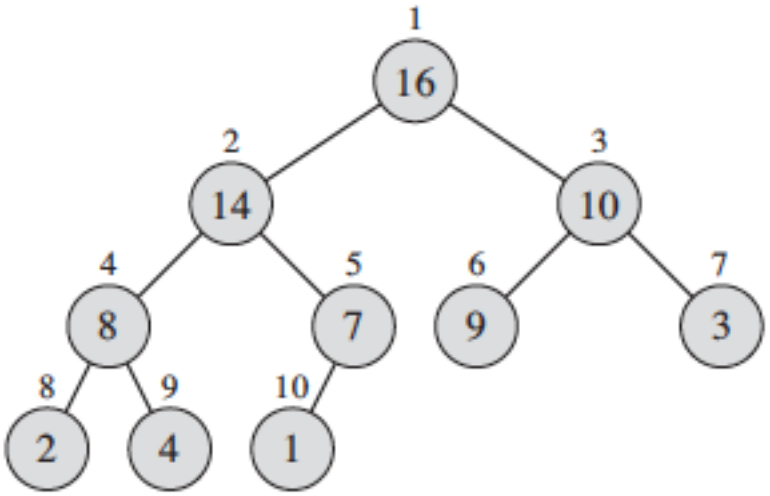
扔鸡蛋问题

- 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

问题1：heap和heapsort

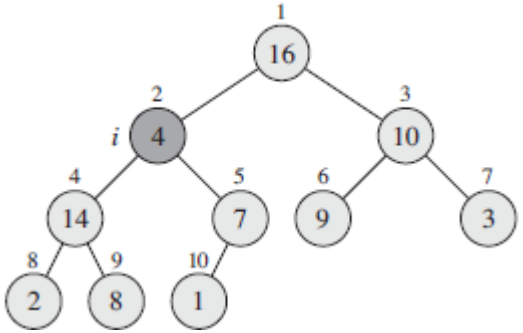
- 什么是堆？
- 它擅长于dynamic set的哪些操作？

Search
Insert
Delete
Minimum
Maximum
Successor
Predecessor



问题1：heap和heapsort (续)

```
MAX-HEAPIFY(A, i)
1  l = LEFT(i)
2  r = RIGHT(i)
3  if l ≤ A.heap-size and A[l] > A[i]
4       largest = l
5  else largest = i
6  if r ≤ A.heap-size and A[r] > A[largest]
7       largest = r
8  if largest ≠ i
9       exchange A[i] with A[largest]
10      MAX-HEAPIFY(A, largest)
```

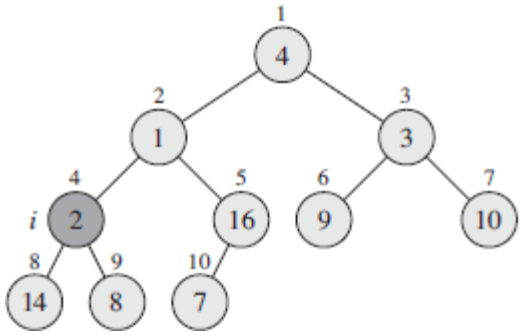


- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 你能解释它的运行时间吗？

$$T(n) \le T(2n/3) + \Theta(1)$$

问题1：heap和heapsort (续)

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

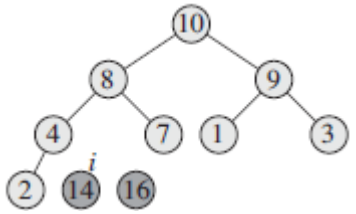


- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 你能解释它的运行时间吗？

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

问题1：heap和heapsort (续)

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

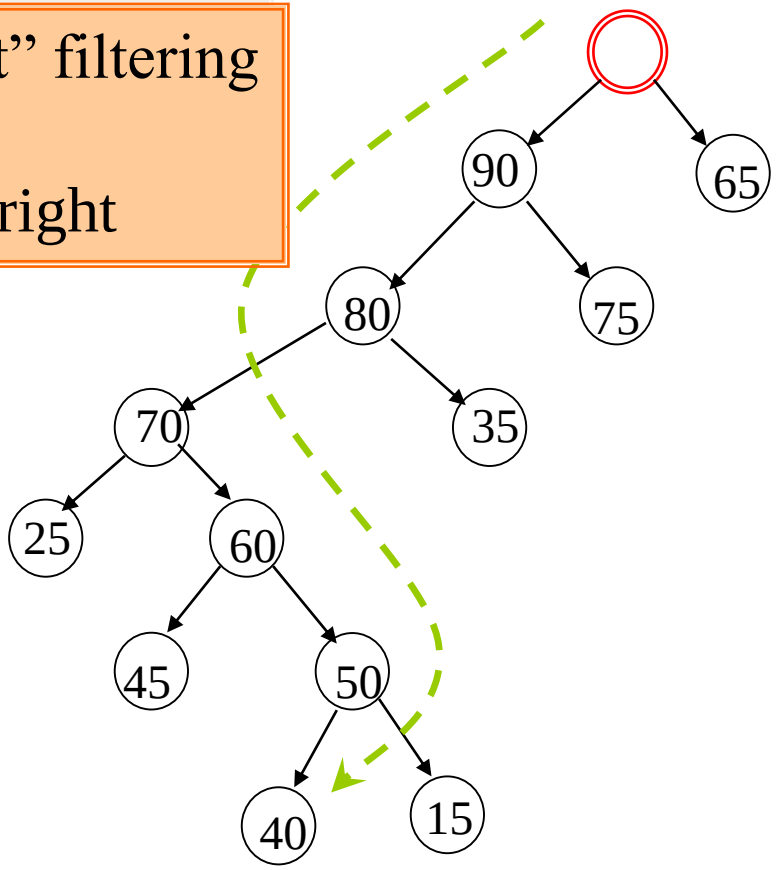


- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 你能解释它的运行时间吗？$O(n \lg n)$

- 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？

HEAPIFY

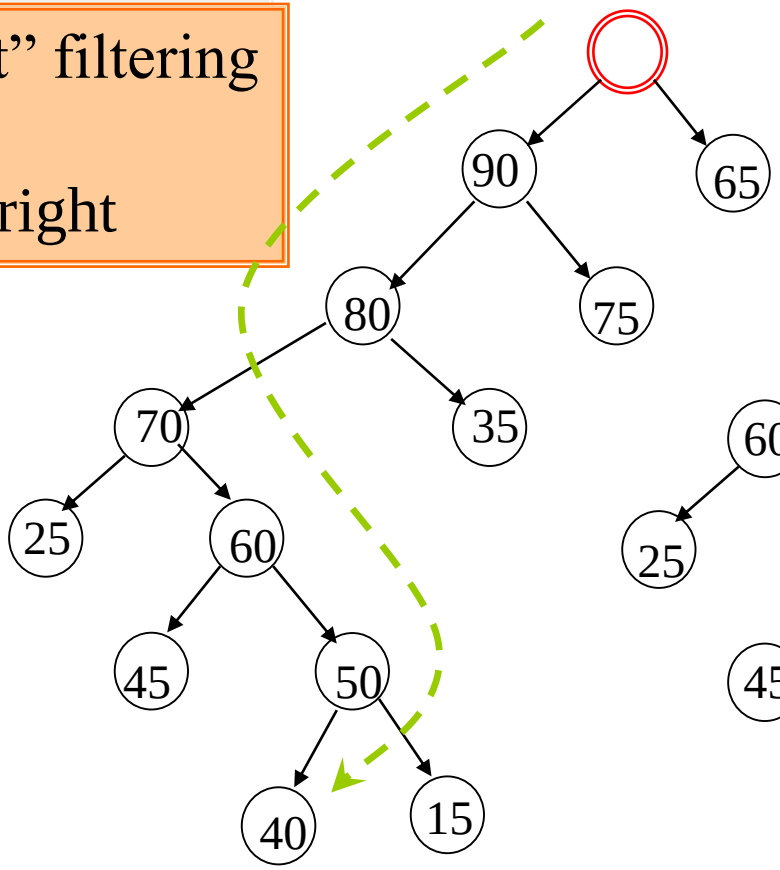
"Vacant" filtering down:
left vs. right

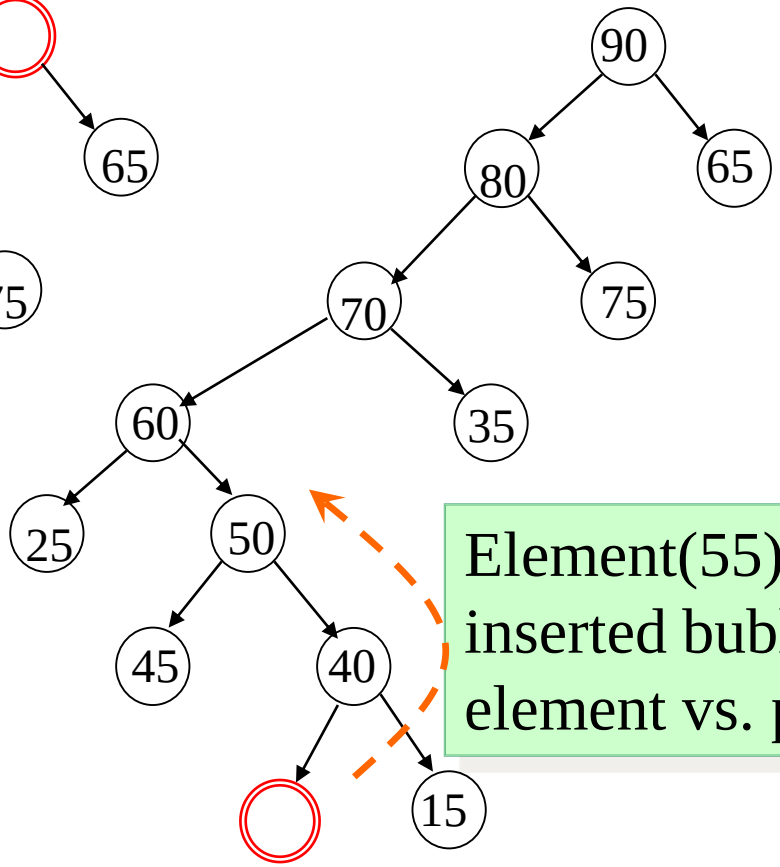


Risky HEAPIFY

In fact, the “risk” is no more than “no improvement”

“Vacant” filtering down:
left vs. right

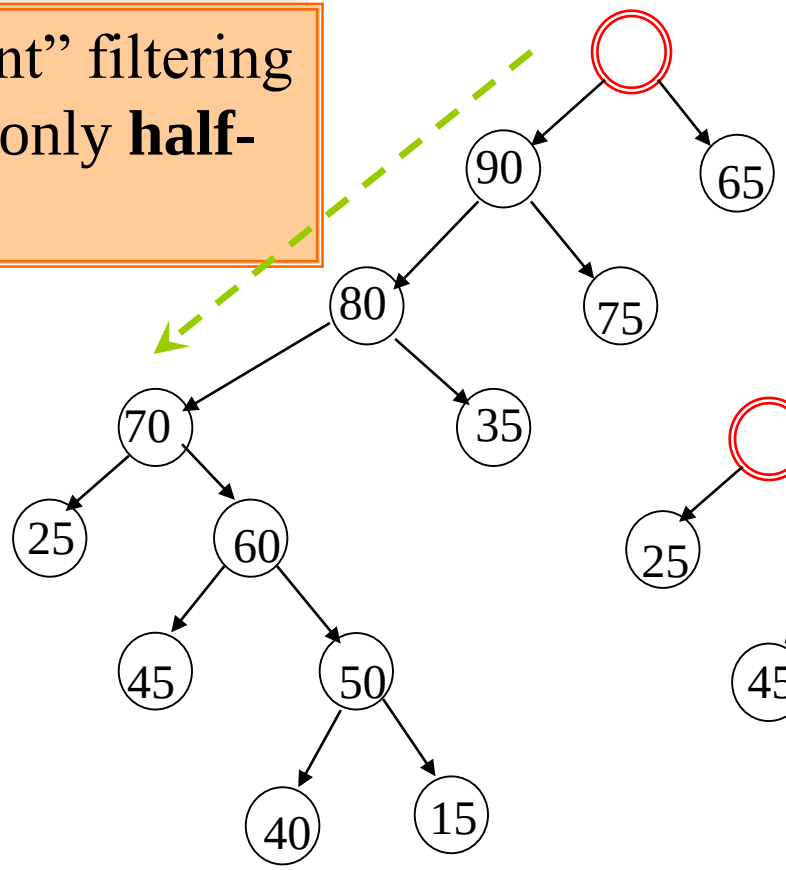


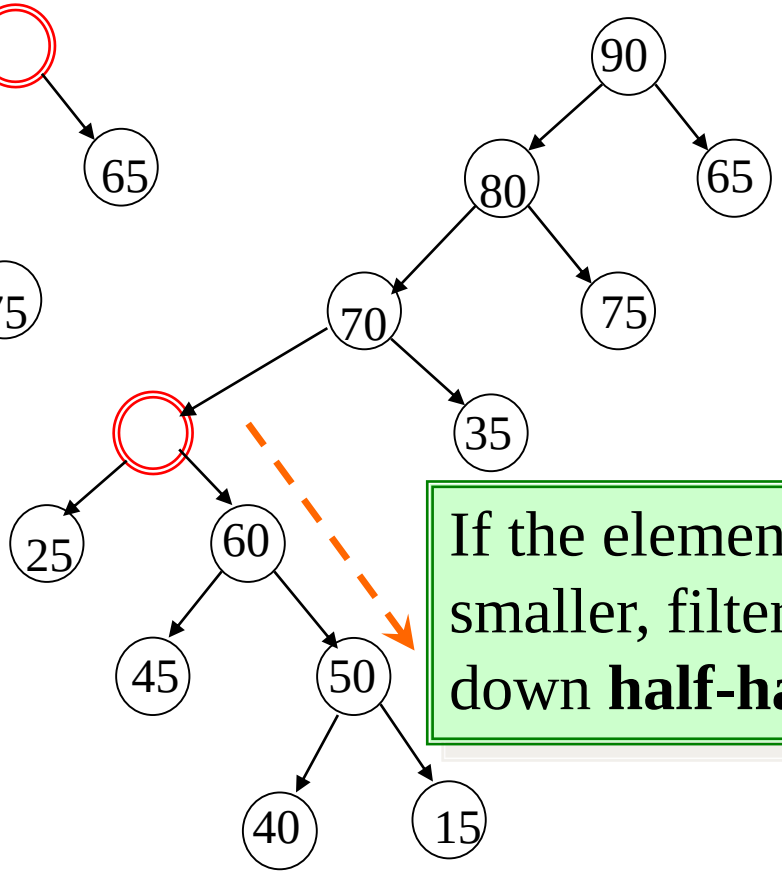


Element(55) to be inserted bubling up:
element vs. parent

Improvement by Divide-and-Conquer

"Vacant" filtering down only **half-way**





If the element is smaller, filtering down **half-half-way**

The bubbling up will not beyond last vacStop

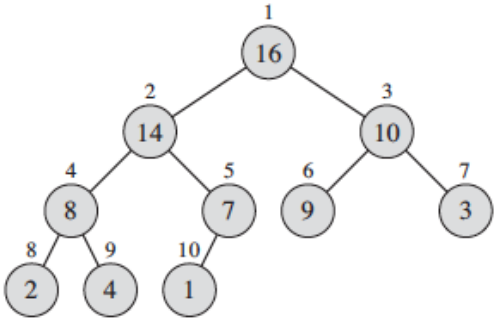
问题2： priority queue

- 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)

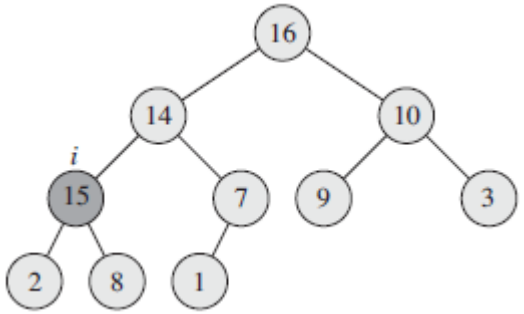
```
HEAP-EXTRACT-MAX(A)
1  if A.heap-size < 1
2        error "heap underflow"
3  max = A[1]
4  A[1] = A[A.heap-size]
5  A.heap-size = A.heap-size − 1
6  MAX-HEAPIFY(A, 1)
7  return max
```



- 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)

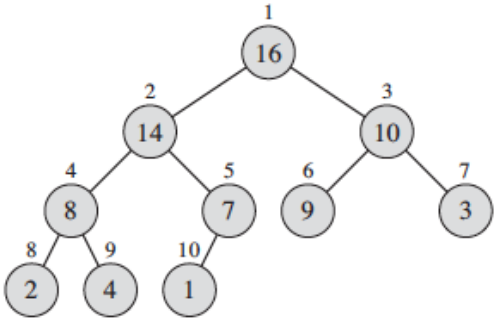
```
HEAP-INCREASE-KEY(A, i, key)
1  if key < A[i]
2      error "new key is smaller than current key"
3  A[i] = key
4  while i > 1 and A[PARENT(i)] < A[i]
5      exchange A[i] with A[PARENT(i)]
6      i = PARENT(i)
```



- 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)

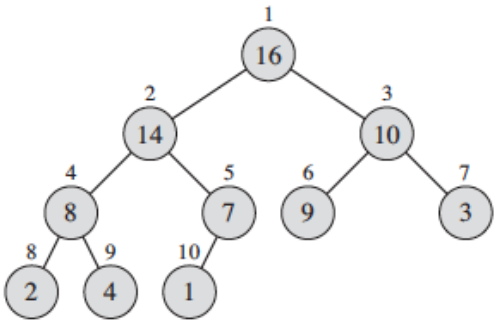
```
MAX-HEAP-INSERT(A, key)
1  A.heap-size = A.heap-size + 1
2  A[A.heap-size] = −∞
3  HEAP-INCREASE-KEY(A, A.heap-size, key)
```



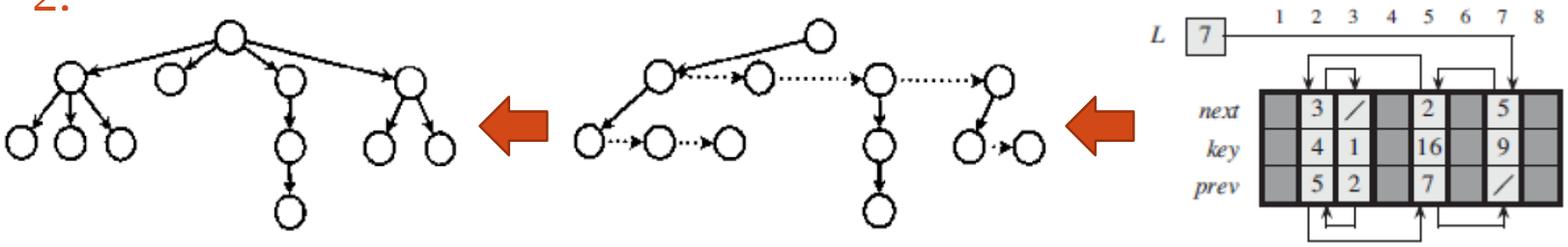
- 这个算法的作用是什么？
- 你能简述它的主要过程吗？
- 你能证明它的正确性吗？
- 你能解释它的运行时间吗？ $O(\lg n)$

问题2： priority queue (续)

- 你能实现O(lg*n*)的HEAP-DELETE(*A*, *i*)吗？



问题3：ADT

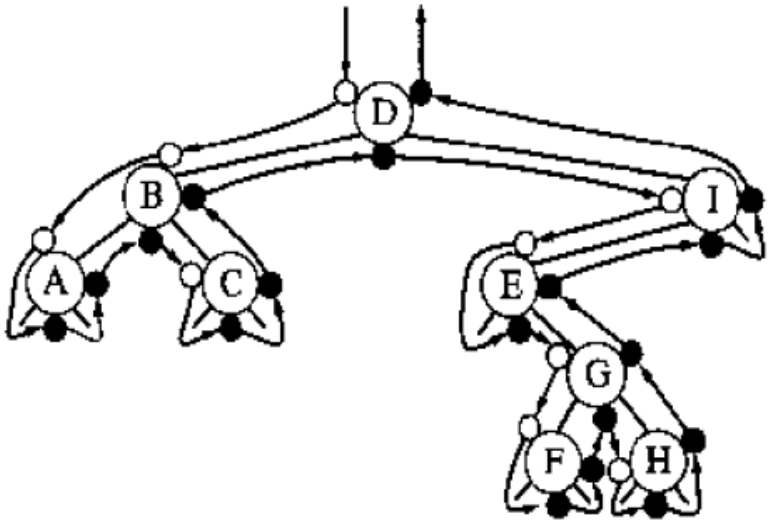
1. priority queue ← heap ← array
2. 

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.

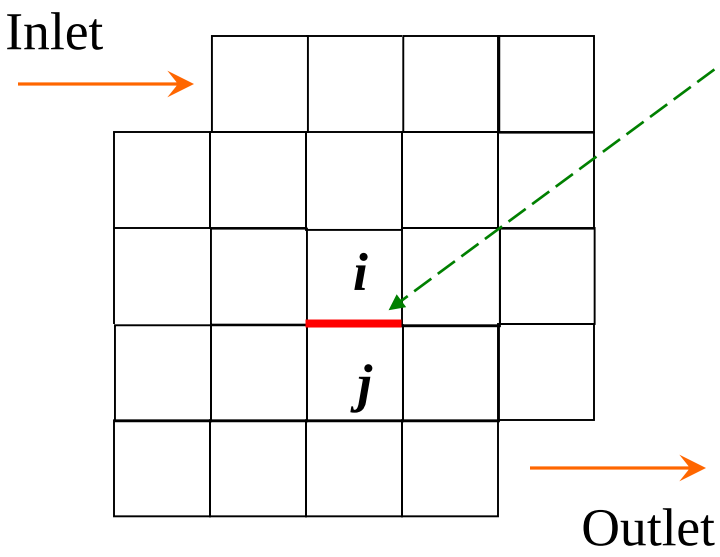
Postconditions: find(sets, *e*) = *e*; that is, *e* is the set id of a singleton set containing *e*.

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.