Data structure [A06]

김종규, PhD

2017-04-10

Heap and tree

- Heap: always left adjusted?

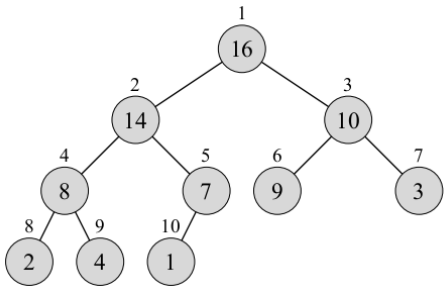


그림: Conceptual heap

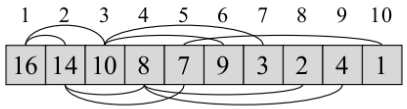


그림: Array implementation

$\longrightarrow$ Heapsort: Buildheap, heapify $\longrightarrow$ Heap 을 구성할 때는 가장 효과적이다.

Outline

- Circular linked list
- Stable sort
- Tree
- Mathematical terminology

Abstract data type: Queue

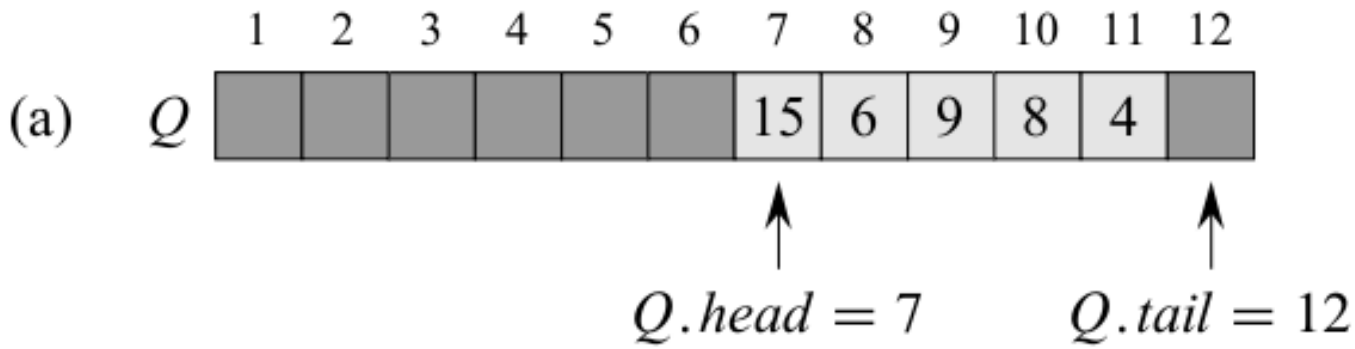


그림: Queue

Queue using list

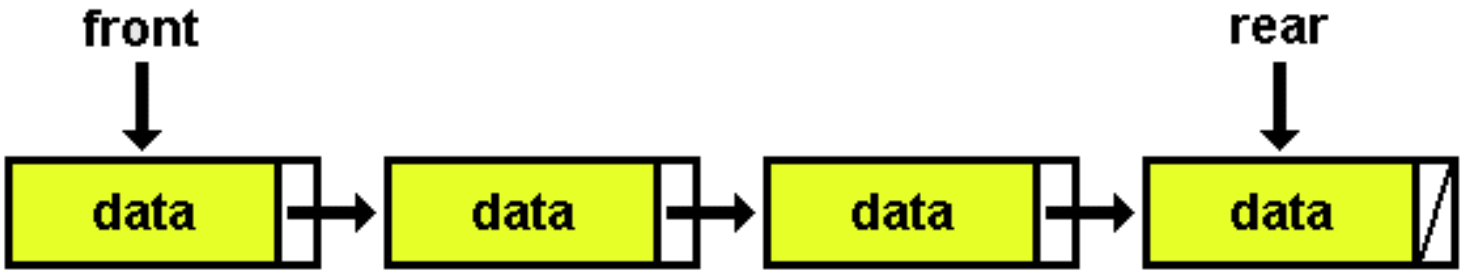


그림: Queue using linked list

Circular linked list

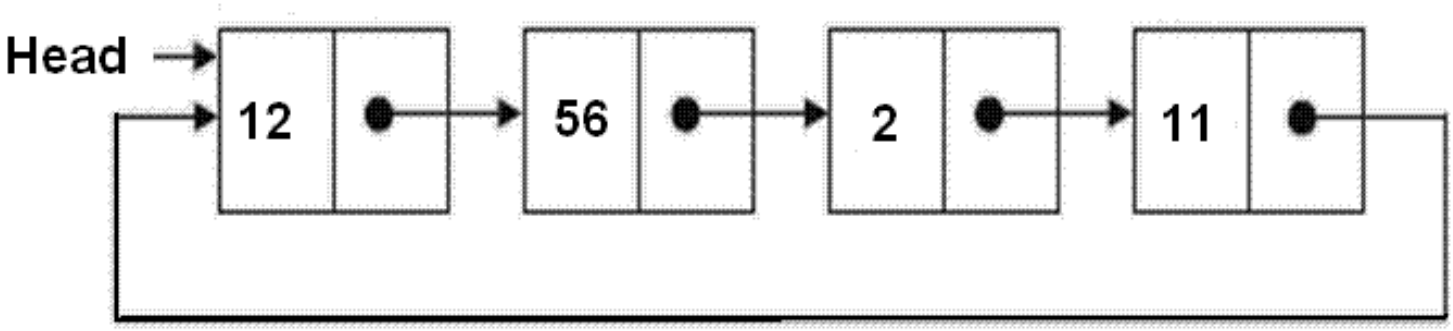


그림: Circular linked list

Unstable sorting

- Sorted by name

```
Name, age
David, 5
Henry, 5
Mary, 5
Tom, 3
```

- Sorted by age $\longrightarrow$ names are not sorted

```
Name, age
Tom, 3
Henry, 5
Mary, 5
David, 5
```

Stable sorting

- Sorted by name

```
Name, age
David, 5
Henry, 5
Mary, 5
Tom, 3
```

- Sorted by age $\longrightarrow$ names remain sorted (stable)

```
Name, age
Tom, 3
David, 5
Henry, 5
Mary, 5
```

- Heapsort has this property!

Review: Heap

- Heap
  - 가장 꼭대기에 가장 큰 값이 올라와 있는 것 (The largest value resides on top)
  - 아래쪽에도 같은 원리가 반복 적용 되는 것 (The same rule applies to lower levels recursively)

Heap: 배열에 저장

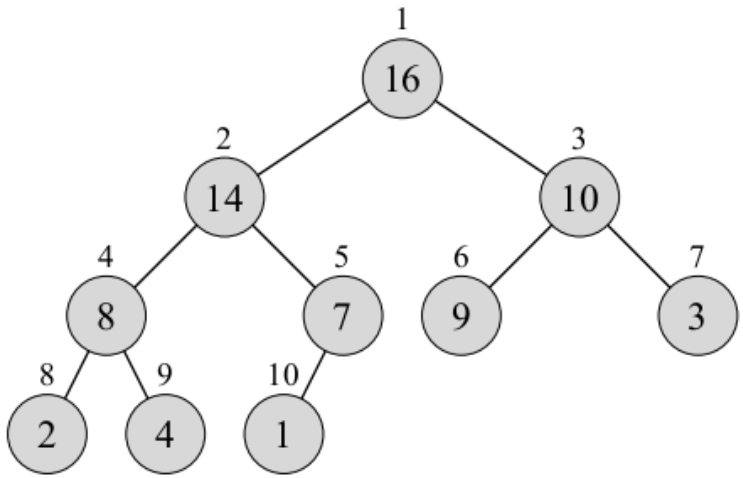


그림: Conceptual heap

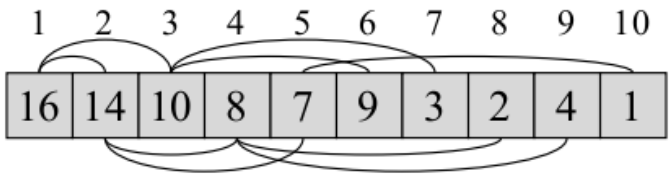


그림: Array implementation

Heap: Basic operations

```
PARENT(i)
1  return ⌊i/2⌋

LEFT(i)
1  return 2i

RIGHT(i)
1  return 2i + 1
```

그림: Basic operations

Heap: linked list 에 저장

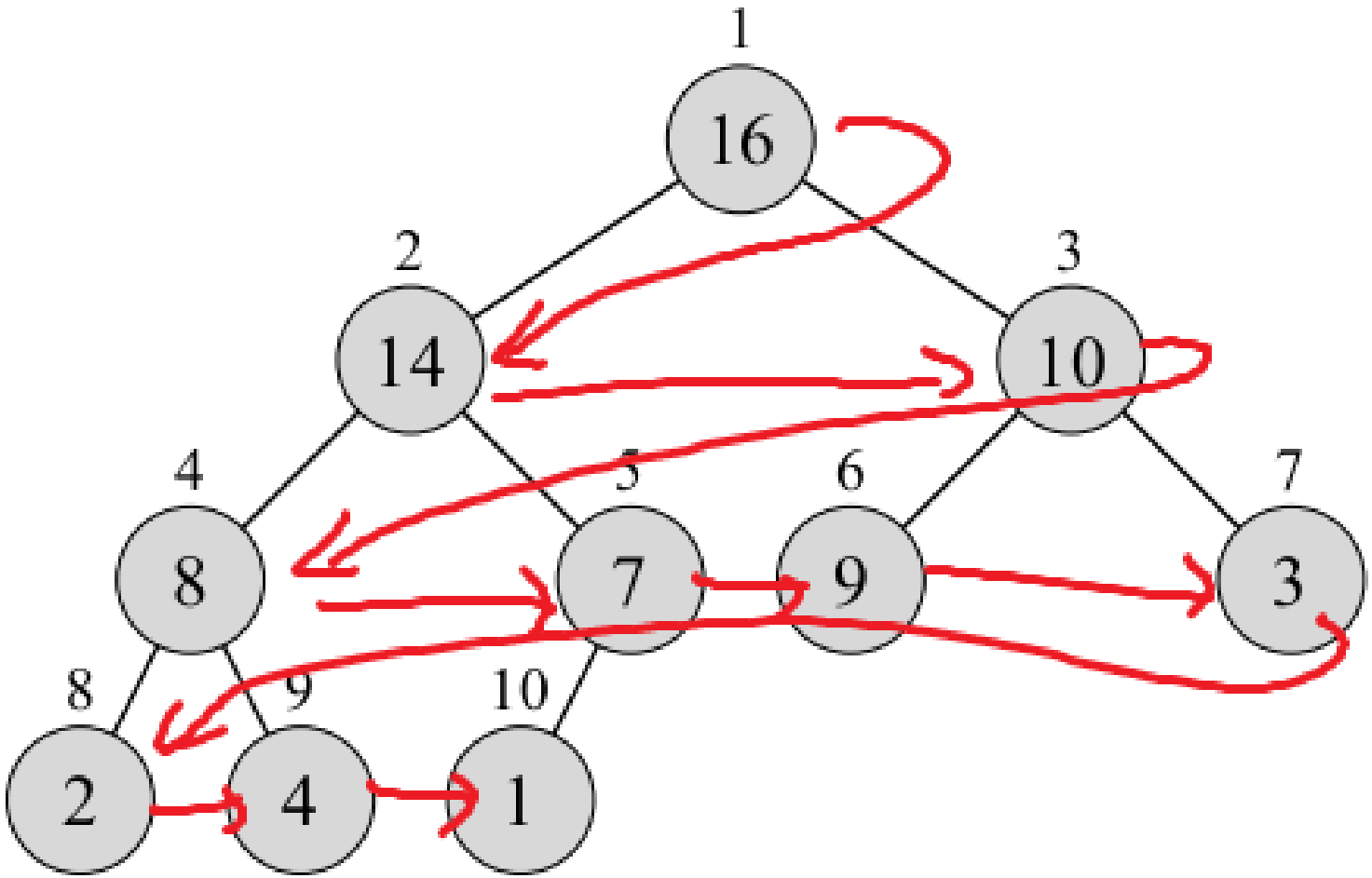


그림: Heap on List

Find n-th element

```
def find_nth(L,n):
  x = L.head
  i = 0
  while x.next != None and i < n:
    x = x.next
    i++
  if i == n:
    return x
  else:
    error("Out of range")
```

- $O(n)$

Search using linked list

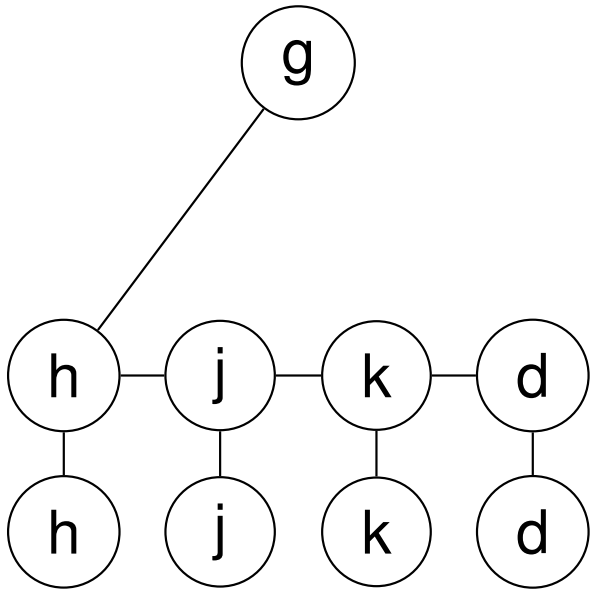
```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
3      x = x.next
4  return x
```

그림: Searching a value

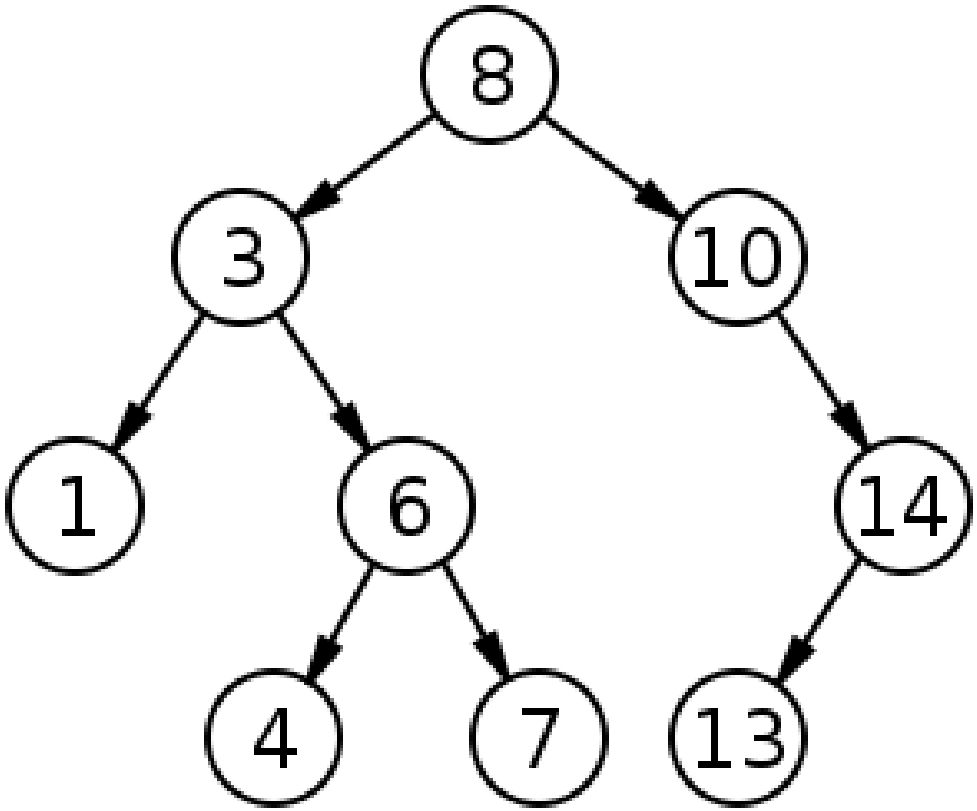
- $O(n)$

Binary tree

- Child 가 많아야 둘인 tree
- 임의의 tree 를 표현할 수 있다



Binary search tree



Tree 에 대한 *재미 없는* 정의 (1/2)

- Tree: 하나의 부모 (parent) 에 0 개 이상의 자손이 연결된 것
  - 부모, 자식 등은 node 로 나타낸다 (partial order)
  - 부모와 자식은 link (혹은 edge) 로 연결된다.
  - 자손은 그 자체로 tree 를 구성한다
  - 자손이 구성한 tree 는 subtree 라고 부른다.
- Rooted tree: tree 의 시작점이 되는 하나의 부모를 중심으로 구성된 tree
  - children 이 없는 rooted tree 의 height 는 1 이다
  - children 이 있는 경우 각각의 children height 을 구하고 이중 최대 값을 찾아 1 을 더한다.

Tree 에 대한 *재미 없는* 정의 (2/2)

- Binary tree: rooted tree 의 일종으로 child 가 많아야 두 개로 제한되며 각각 left child, right child 로 부른다.
- Binary search tree: binary tree 의 일종으로 왼쪽에는 parent 보다 작은 값들이 존재하고 오른쪽에는 parent 보다 큰 값이 존재하는 조건이 모든 tree 의 모든 node 에 대해서 적용된다
- Balanced tree: binary search tree 의 일종으로 left child 의 height 와 right child 의 height 의 차이가 크지 않다

Tree 의 *흥미로운* 응용

- 메모리의 제약을 받지 않고 얼마든지 데이터를 저장할 수 있다.
- $O(\log n)$ 의 시간 복잡도로 원하는 값을 찾을 수 있다.
- $O(n \log n)$ 의 시간 복잡도로 주어진 데이터를 정렬 (sorting) 할 수 있다
- 정보를 체계적으로 저장할 수 있다 $\longrightarrow$ 운영체제의 디렉토리 구조

Tree algorithms (1/3)

- Tree: 하나의 부모 (parent) 에 0 개 이상의 자손이 연결된 것
  - Tree 를 pseudocode 의 형태로 정의하고 새로운 child 를 추가하는 연산 addChild 를 정의하시오

```
class TreeNode:
    def __init__(self):
        self.val = 0
        self.children = None
        self.next = None
    def addChild(self,child):
        if self.children:
            child.next = self.children
            self.children = child
        else:
            self.children = child
```

Tree algorithm (2/3)

- Tree 의 정의 (C 언어)

```
struct TreeNode {
  int val;
  struct TreeNode* children;
  struct TreeNode* next;
};
typedef struct TreeNode* TreeNodePtr;
TreeNodePtr alloc_node(int val) {
  TreeNodePtr n = ( TreeNodePtr )malloc(sizeof(struct TreeNode));
  n->val = val;
  n->children = NULL;
  n->next = NULL;
  return n;
}
```

Tree algorithm (3/3)

- 0 개 이상의 children 을 갖는 tree 를 정의하고 주어진 tree 의 내용을 출력하시오

```
def printTree(tree, level):
    node = tree
    while node:
        for i in range(level):
            print("    ", end="")
        print (node.val)
        printTree(node.children,level+1)
        node = node.next
```

Rooted tree algorithm

- Rooted tree 를 pseudocode 의 형태로 정의하시오.

  $\longrightarrow$ Tree 와 동일
- Rooted tree 의 height 를 구하시오

```
def heightTree(tree):
    if tree.children is None:
        return 1
    child = tree.children
    height = 0
    while child:
        if height < heightTree(child):
            height = heightTree(child)
        child = child.next
    return height + 1
```

Binary tree algorithm

- Binary tree 를 pseudocode 의 형태로 정의하시오.
- Binary tree 의 height 를 구하시오.

```
def bt_height(tree):
    if tree is None:
        return 0
    lh = bt_height(tree.left)
    rh = bt_height(tree.right)
    return max(lh, rh) + 1
```

Administrivia

- Quiz #02 4/12 (Wed)
- 중간고사 예행 연습
  - 지난 번 퀴즈보다 난이도 높음

Comparison: List v. Tree

	List	Tree
Memory	dynamic	dynamic
ADT	insert, delete, find	insert, delete, find
Average	$O(1), O(1), O(n)$	$O(\log n), O(\log n), O(\log n)$
Worst	$O(1), O(1), O(n)$	$O(n), O(n), O(n)$
Traversal	Iterative	Recursive

- 최악의 경우에도 $O(\log n)$ 을 보장하도록 할 수 있는 Tree? $\longrightarrow$ Balanced tree (e.g., Red-black tree)
- Average 에 $O(\log n)$ 을 보장하는 List? $\longrightarrow$ skip list (not for this semester)

Review: Set

- Set: A collection of elements
- $A \subset B \vee A = B \equiv A \subseteq B$
  - $A \subset B$: A is a proper subset of B
- Laws of sets
  - Empty set laws: $A \cap \phi = \phi$, $A \cup \phi = \phi$
  - Idempotency laws: $A \cap A = A$, $A \cup A = A$
  - Commutative laws: $A \cap B = B \cap A, \cdots$
  - Distributive laws: $A \cap (B \cup C) = (A \cap B) \cup (A \cap C) \cdots$
  - Absorption laws: $A \cap (A \cup B) = A \cup B, \cdots$
  - DeMorgan's laws: $A - (B \cap C) = (A - B) \cup (A - C)$, $\cdots$

Relation

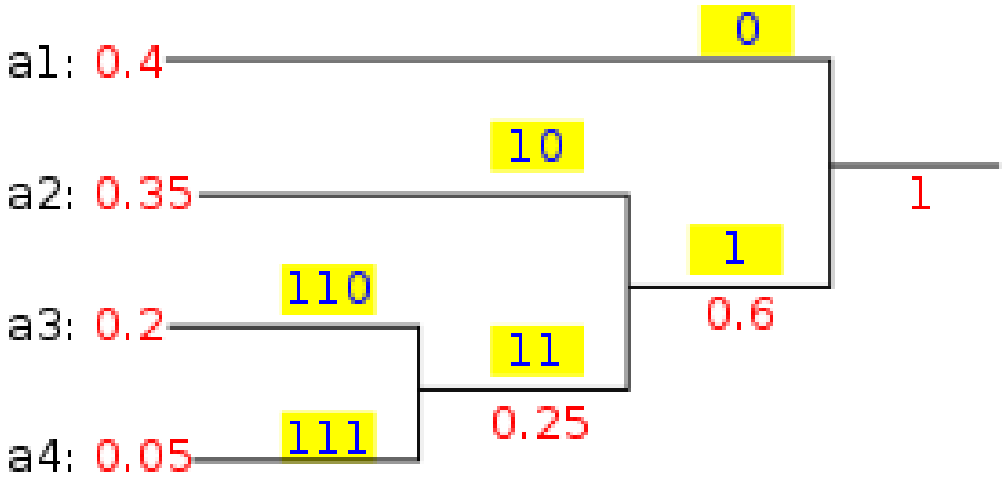
- Binary relation R on two sets A and B: $R \subseteq A \times B$

  $aRb \equiv (a, b) \in R$
- Binary relation R on a set A: $R \subseteq A \times A$
- Reflexive: $\forall_{a \in A}(a, a) \in R$
- Symmetric: $\forall_{(a,b) \in R}(b, a) \in R$
- Antisymmetric: $aRb \wedge bRa \rightarrow a = b$
- Transitive: $\forall_{(a,b),(b,c) \in R}(a, c) \in R$

More on relation

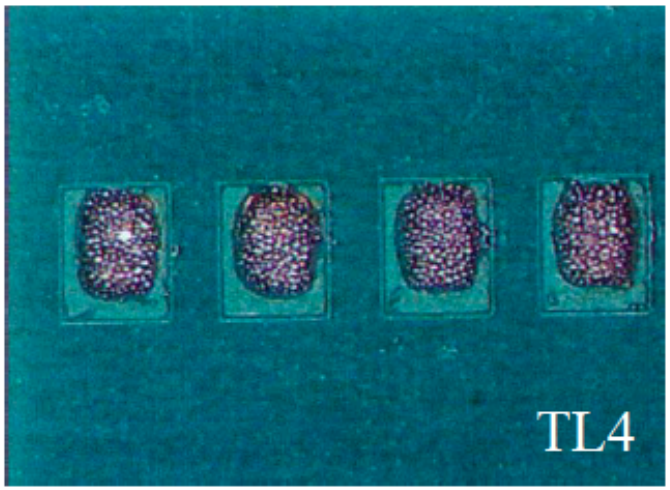
- Equivalence: Reflexive, transitive, symmetric
- (Partial) order: Reflexive, transitive, antisymmetric
- Total order: Partial order and $\forall_{a,b} aRb \vee bRa$

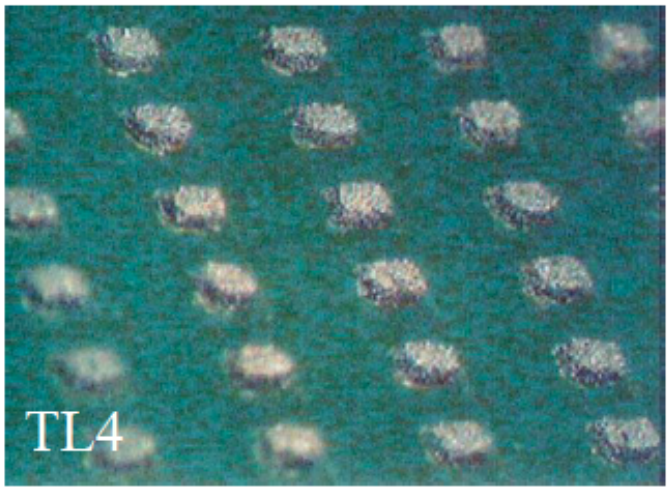
Data compression (Huffman coding)

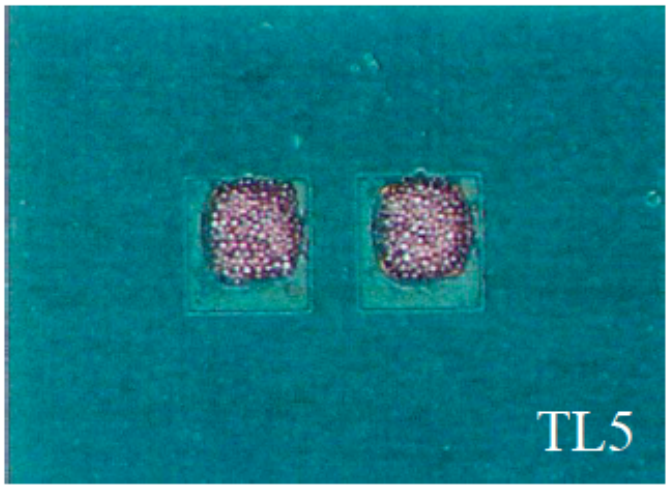


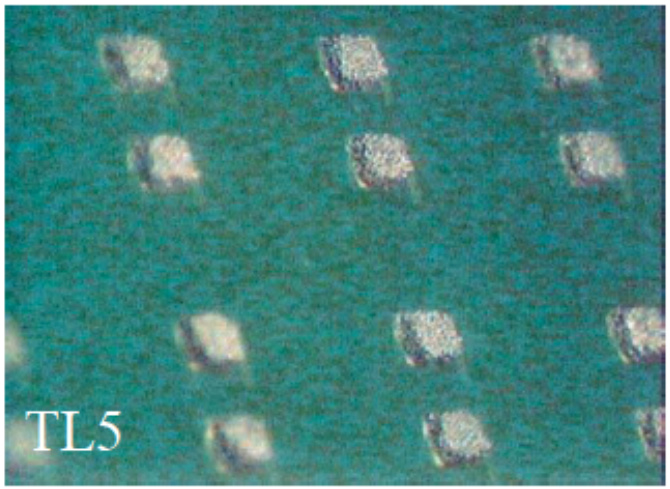
Practical applications

- Good product



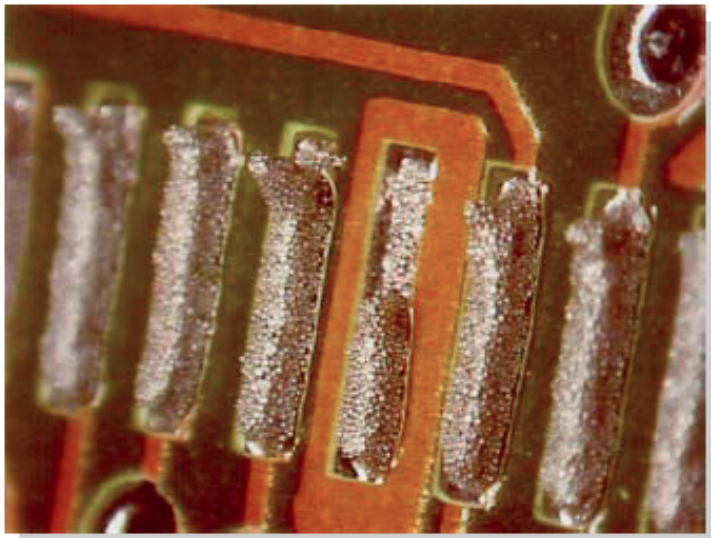


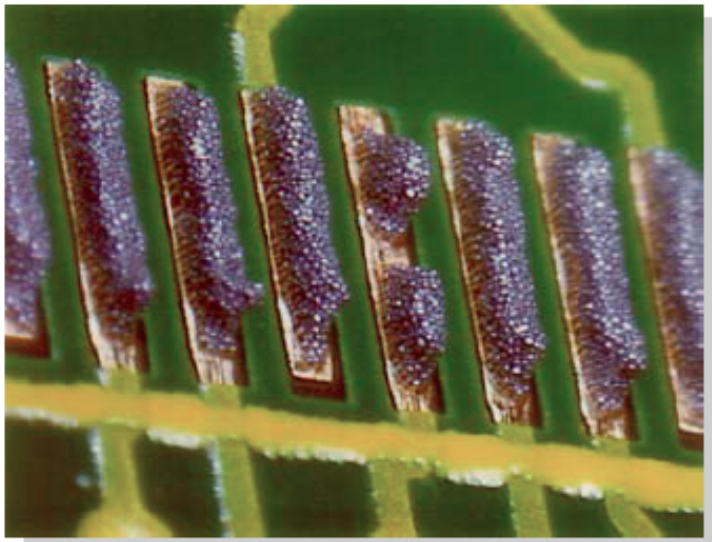




Practical applications

- Bad product





Wrap-up

- A tree is an abstract data structure which has many interesting characteristics, i.e., useful in building softwares.
- Binary search trees are a special kind of tree which has at most two children.
- Data stured in a binary search tree can be retrieved in $O(\log n)$ with a recursive algorithm
- Simple binary search trees can be detoriorated with insertions and deletions, hence we will learn balanced tree