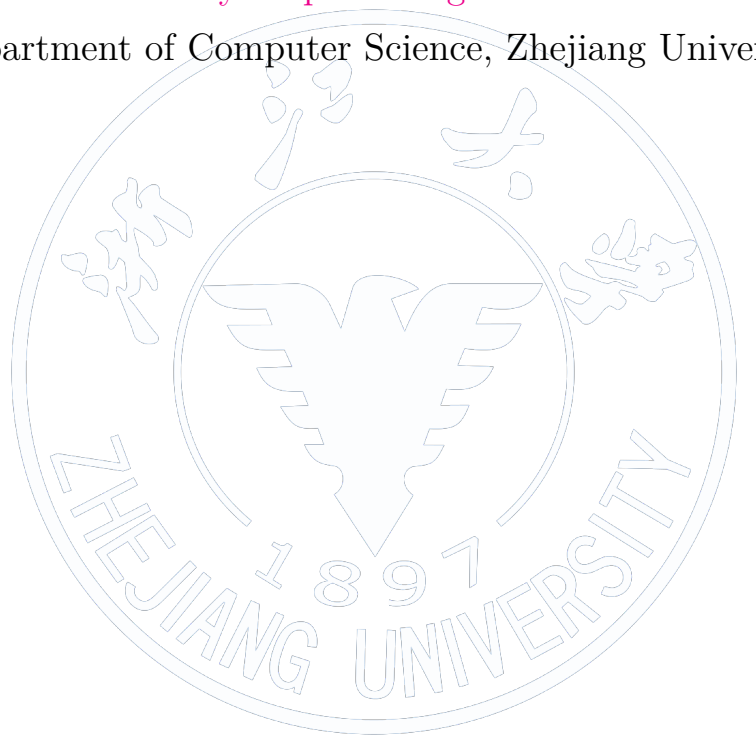
資料結構

Data Structure

TZU-CHUN HSU[1]

[1]vm3y3rmp40719@gmail.com

[1]Department of Computer Science, Zhejiang University



2022 年 3 月 12 日

Version 5.0.0

Disclaimer

本文「資料結構」為台灣研究所考試入學的「資料結構」考科使用，內容主要參考 Introduction to Algorithms[1]，以及 wjungle 網友在 PTT 論壇上提供的資料結構筆記 [2]。本文作者為 TZU-CHUN HSU，本文及其 LaTeX 相關程式碼採用 **MIT 協議**，更多內容請訪問作者之 GitHub 分頁Oscarshu0719。

1 Overview

1. 本文頁碼標記依照 TKB 筆記 [2] 的頁碼。

2. TKB 筆記 [2] 章節頁碼：

Chapter	Page No.	Importance
1	3	⋆⋆⋆
2	259	⋆
3	52	⋆⋆⋆
4	259	⋆
5	82	⋆⋆⋆⋆⋆
6	228	⋆⋆⋆⋆
7	180	⋆⋆⋆⋆
8	221	⋆⋆⋆
9	129	⋆⋆⋆⋆

3.

$$\begin{aligned} \log 2 &= 0.3010 \\ \log 3 &= 0.4771 \\ \log 5 &= 0.6990 \\ \log 7 &= 0.8451 \end{aligned} \tag{1}$$

4. OBST 在「演算法」中，不再贅述。

Trees				
Tree	Insert x	Delete x	Search x	Remark
BST	$O(\log n) \sim O(n)$			Create: $O(n \log n) \sim O(n^2)$
AVL tree	$O(\log_m n)$			$F_{h+2} - 1 \leq n \leq 2^h - 1$
B tree				$1 + 2\frac{\lceil \frac{m}{2} \rceil^{h-1} - 1}{\lceil \frac{m}{2} \rceil - 1} \leq n \leq \frac{m^h - 1}{m-1}$
RBT				$h \leq 2\log(n+1)$
Splay tree				Worst: $O(n)$, Amortized: $O(\log n)$

Priority queues					
Operations	Max (Min)	Min-max & Deap & SMMH	Leftist	Binomial	Fibonacci
Insert x	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n), O(1)^*$	$O(1)^*$
Delete max	$O(\log n)$	$O(\log n)$			
Delete min	$O(n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)^*$
Delete x				$O(\log n)$	$O(\log n)^*$
Merge	$O(n)$		$O(\log n)$	$O(\log n)$	$O(1)^*$
Decrease key				$O(\log n)$	$O(1)^*$
Search x	$O(n)$				
Find max	$O(1)$	$O(1)$			
Find min		$O(1)$		$O(\log n)$	$O(1)$
Remark			$shortest$(root) $\leq \log(n+1) - 1$		

Sorting algorithms					
Method	Time complexity			Space complexity	Stable
	Best	Worst	Average		
Insertion	$O(n)$	$O(n^2)$		$O(1)$	$\surd$
Selection	$O(n^2)$			$O(1)$	$\times$
Bubble	$O(n)$	$O(n^2)$		$O(1)$	$\surd$
Shell	$O(n^{1.5})$	$O(n^2)$		$O(1)$	$\times$
Quick	$O(n \log n)$	$O(n^2)$	$O(n \log n)$	$O(\log n) \sim O(n)$	$\times$
Merge	$O(n \log n)$			$O(n)$	$\surd$
Heap	$O(n \log n)$			$O(1)$	$\times$
LSD Radix	$O(n \times k)$			$O(n+k)$	$\surd$
Bucket/MSD Radix	$O(n)$	$O(n^2)$	$O(n+k)$	$O(n \times k)$	$\surd$
Counting	$O(n+k)$				$\surd$

2 Summary

1. **Theorem (17)** Permutation：

```
1: function PERM(list, i, n)
2:     if i == n then
3:         PRINT(list)
4:     else
5:         for j := i to n do
6:             SWAP(list, i, j)
7:             PERM(list, i + 1, n)
8:             SWAP(list, i, j)
9:         end for
10:    end if
11: end function
```

2. **Theorem (87)** 節點數：

$$n = (\sum_{i=1}^{\deg} i \times n_i) + 1 \tag{2}$$

$$n_0 = n_2 + 1\text{（二叉樹）}$$

3. **Theorem (116)**

```
1: function CreateMinHeap(Tree s, size n)
2:     for i := n/2 to 1 do                          ▷ Start from parent of the last node.
3:         tmp := s[i]
4:         j := 2 × i                                 ▷ Left child of i.
5:         while j ≤ n do                             ▷ There is a child.
6:             if j < n then                          ▷ Right child exists.
7:                 if s[j] > s[j + 1] then            ▷ Choose the smaller child.
8:                     j := j + 1
9:                 end if
10:            end if
11:            if tmp ≤ s[j] then
12:                Break.
13:            else                                   ▷ Percolate one level.
14:                s[j/2] := s[j]
15:                j := j × 2
16:            end if
17:        end while
18:        s[j/2] := tmp
19:    end for
20: end function
```

4. **Theorem (257)** 尋找 articulation point：若 root 有 ≥ 2 子節點，則 root 為 articulation point；$\exists$ 非 root 節點 u，若 $\exists v$ 為 u 子節點，且 $low(v) \geq dfn(u)$，則 u 為 articulation point。

5. **Theorem ()**

   - AVL trees are ideal for sorting items of an **ordered dictionary**.
   - Different number binary trees of height h:

$$H_n = \begin{cases} 2H_{n-1} \times \sum_{i=0}^{h-2} H_i + H_{i-1}^2 & , h \geq 2 \\ H_0 = 1, H_1 = 3 \end{cases} \tag{3}$$

   - *vector* in C++ STL is similar to regular array that it supports direct access.
     - Elements in *vector* is stored in consecutive memory.
     - If we make a sequence of n insertions, there is a way that we can achieve $\Theta(n)$ amortizeed cost.
   - (105NYCU-16) We can use **array** to implement **any** data structure, even user-defined structure.

- (110NTU-1) (**FALSE**) Quicksort where the leftmost or rightmost element is always chosen as pivot CAN the worst-case occur.

References

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3 edition, 2009.

[2] wjungle@ptt. 資料結構 @tkb 筆記. https://drive.google.com/file/d/0B8-2o6L73Q2VeFpGejlYRk1WeFk/view?usp=sharing, 2017.