# My Notes

Important Concepts worth keeping

Today: / /

**Queue** * First In First Out (FIFO), First Come First Serve (FCFS)

- 頭 front, 尾 rear
- Ex: 排隊買票, 列印排程.
- ADT Queue operation: ①建構 ②解構 ③是否為空 ④新增 ⑤移除 ⑥檢索

  isEmpty(), enqueue(ItemType), dequeue(), getFront() dequeue()
- Reading a string of characters:

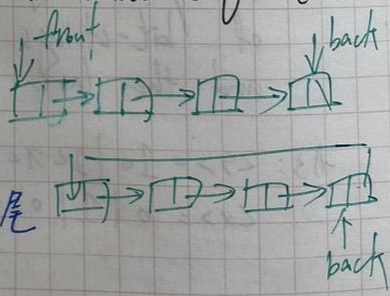
```
aQueue.createQueue()
While (not end of line){
    Read a new ch;
    aQueue.enqueue(ch);
}
```

a sequence digit to decimal value

eg. 24 = (2*10+4)*10+7

```
do { aQueue.dequeue(ch)
   } while (ch is blank)
n=0, done=FALSE
while (!done && ch is digit){
    n=n*10+(ch+'0')
    if (aQueue.isEmpty()) done=TRUE
    else aQueue.dequeue(ch)
```

- Recognizing palindromes (e.g., add, mom, noon, civic)
  - ↳ stack: reverse the order
  - ↳ Queue: preserve the order
- Implements of the ADT Queue
  - array-based (circular array)
  - pointer-based
    - linear - 1個, 1尾
    - circular - 1尾







A word fitly spoken is like apples of gold in baskets of silver.
《Proverbs》

## My Questions
Problems & Difficulties needing exploration

- Array, Pointer 比較
  - 靜、動態空間
  - Pointer 效率較高
- 應用：Simulation 模擬
  - ↳ 行為模型
  - ↳ 目的：generate statistics（統計）、predict（預測）
  - ↳ 例子：bank teller（銀行辦事員）
    進來時間、等待時間、辦事時間
  - 時間推動

| | Arrival | duration | Departure | waiting |
|---|---|---|---|---|
| (時間軸) | 20 | 5 | 25 | 0 |
| time of next event | 22 | 4 | 29 | 3 |
| | 23 | 2 | 31 | 6 |
| | 30 | 3 | 34 | 1 |

(Events input)
AWT = 10/4 = 2.5
MWT = 6
MQL = 2 排長
AQL = 1+2+1/4 = 1

## My Opinions
Thoughts, inspirations, and suggestions

- bank simulation - Arrival : Exteran 輸入決定時間
  - Departure : interval 模擬時間
- Bank simulation is event-driven 需建立事件清單

simulate()
Create an empty bankQueue
Create an empty eventList
Get the earliest arrival event x from Input file
Put x into eventList

# My Notes

Important Concepts worth keeping

Today : / /

- 單一佇列 Events (input file)

| Arriva | transaction | Departure | waiting |
|---|---|---|---|
| 5 | 9 | 14 | 0 |
| 7 | 5 | 19 | 7 |
| 14 | 5 | 24 | 5 |
| 30 | 5 | 35 | 0 |
| 32 | 5 | 40 | 3 |
| 34 | 5 | 45 | 6 |
| 38 | 3 | 48 | 7 |

- 三種系統架構
  - Single teller / single queue
  - Multiple teller / single queue
  - Multiple teller / Multiple queue

- 多重佇列 (two queues)

Events (input file)

| Arriva | transaction |
|---|---|
| 5 | 9 |
| 7 | 5 |
| 14 | 5 |
| 30 | 5 |
| 32 | 5 |
| 34 | 5 |
| 38 | 3 |

* Big O
1. O(1) constant
2. O(log2n) logarithmic
3. O(n) linear
4. O(nlog2n)
5. O($n^2$)
6. O($n^3$)
7. O($2^n$)

If you don't know where you're going it doesn't matter what path you take.
– Lewis Carroll

My Notes
Important Concepts worth keeping

Today: , ,

## Bubble Sort 氣泡排序

- 從最右(最左)兩兩比較，若是小→大 or 大→小
- 假設有6筆資料，每回合要比5次
- 概念：當做完一回合，確保最小(大)資料放在正確位置
- 每回合比較次數，遞減，越比越少

## Selection Sort 選擇排序

- 從左到右看一次，選擇一個最小(大)的記錄下來，搬到第一個
- 只需要一次swap，比較次數與bubble sort一樣
- 假設6筆資料，每回合比5次，共做5回合
- 每回合比較次數遞減，越比越少

## Quick Sort

- 類似二次搜尋法(不是等分)
- 選擇一個樞紐(pivot)直接將pivot視為已在正確位置
- 左：item < pivot 右：item ≥ pivot
- 先：分組(耗花時間)→後：遞迴呼叫

Whenever it feels uncomfortable to tell the truth,
that's often the most important time to tell it.
–Jennifer Lopez

## My Questions
Problems & Difficulties needing exploration

Merge Sort 合併排序
- 實用方法(可內部排序也可外部)
- 先：分組 → 各自排序 → 後：合併 (較花時間)
- 遞迴寫法 * 遞迴呼叫次數
- 類似二元搜尋法(等分)
- 比較次數為分組個數相加後再減-(資料總數-1)

Radix Sort 基數排序
- 速度最快
- 需要進位系統，字串也可用
- 排序key可為數字or字串
- 無比較，只有資料搬動

## My Opinions
Thoughts, inspirations, and suggestions

- 分解取部分值 → 分配至對應容器

Comparison of Sort Algorithms

| | Worst case | Average case |
|---|---|---|
| Selection | $n^2$ | $n^2$ |
| Bubble | $n^2$ | $n^2$ |
| Insertion | $n^2$ | $n^2$ |
| Merge | $n \log n$ | $n \log n$ |
| Quick | $n^2$ | $n \log n$ |
| Radix | $n$ | $n$ |

# Trees 樹狀結構.

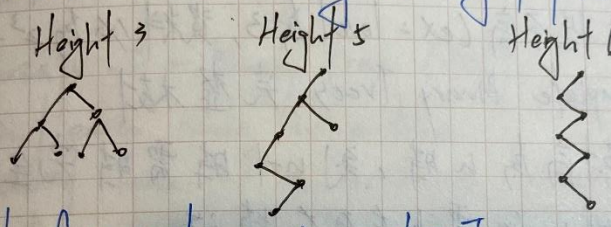
特点：二維資料（較複雜的關係）→ 階級，直屬去關係

- Data-Management Operations
  - ↳ 一般（General）：(i) Insert data into a data collection
    (ii) Delete data from a data collection
    (iii) Ask question about the data in it
  - ↳ 位置導向：處理 $i^{th}$ position
    Example = list, stack, queue, binary tree
  - ↳ 內容導向：根據 data 的值放進正確的位置.
    Example = sorted list, binary search tree

術語：i) Trees are composed of nodes and edges
ii) Trees are hierarchical（階層）
↳ parent-child 親子關係
↳ Ancestor-descendant 祖孫關係
iii) Subtree = any node and its descendants
iiii) root = the highest (A)
v) Leaf = A node with no children
vi) Siblings = nodes with a common parent
vii) Ancestor = a node on the path from root
viii) Descendant = a node on a path from to B
B to a leaf

## My Questions
Problems & Difficulties needing exploration

- Binary tree = 二元樹
  定義：每個節點的分支 <= 2，最多二
- Height
  - number of nodes along the longest path from root to a leaf




- Level of a node n in a tree T
  - n is a root of T, it is at level 1
  - n is not the root of T, its level is 1 greater than the level of its parent

## My Opinions
Thoughts, inspirations, and suggestions

- Height of a tree T defined in term of the levels of its nodes
  - T is empty, height is 0
  - T is not empty, 最大階層 == 樹高
- 樹高的遞迴定義：
  if T is empty, its height is 0
  else height(T) = 1 + max{height($T_L$); height($T_R$)}

# My Notes

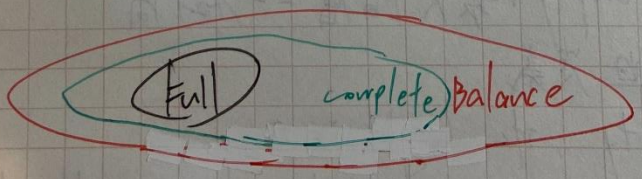
Important Concepts worth keeping

Today: / /

- Type I full Binary Trees 完全樹
  - 定義：A binary tree of height h is full if
    - Nodes at levels ≤ h have two children each

(資料皆固定)

- 限制多，實用性不高 (ex：level 為 3，資料必為 $2^3+1=7$)
- Type II Complete Binary Trees 完整樹
  - 定義 1：
    - 樹高為 n 時，到 n-1 時需時完全樹
    - level n 必需由左至右填滿
  - 定義 2：
    - 樹高為 n 時，到 n-2 時每層皆有 2 個孩子
    - 樹高為 n-1 時，孩子靠左
- Type III Balanced Binary Trees 平衡樹
  - 定義：每個節點的左，右子樹樹高差距不超過 1
    - ↳ 可以不必填滿，也不必靠左

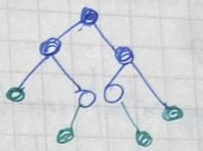



There is no disgrace in honest failure; there is disgrace in fearing to fail.

– Henry Ford

## My Questions
Problems & Difficulties needing exploration

- 二元树性质



$N_2$ = 3, nodes with two children
$N_0$ = 4 Leaves — $N_0 = N_2 + 1$
B = 8 Branches (edges) — $B = |E| = 2 * N_2 + 1 * N_1$ 点跟边差一个

- Traversals of a Binary tree (visit)
  - Visit each node in a tree
  - General form of a recursive traversal algorithm

```
traverse ( in binTree : BinaryTree)
    if (binTree is not empty ) {
        traverse (left subtree of bintree's root)
        traverse (right subtree of bintree's root)
    }
```

前 →
中 →
後 →

## My Opinions
Thoughts, inspirations, and suggestions

- Pre order — visit root before visiting its subtrees
- Inorder — visit root between visting its subtrees
- Postorder — visit root after visting its subtrees

# My Notes

Important Concepts worth keeping

Today : / /

Big O

- Sequential search worst = O(n) Average = O(n) Best = O(1)
- Binary search worst = O(log n)

Stable vs - Unstable

Stable: bubble, insertion, merge, radix

Unstable: selection, quick, heap