資料結構

No.
Date: / /

✳ Queue - First In First Out [FIFO] - First Come First Served (FCFS)

=> New item enter at the back or rear 後端

=> front 前端 [放進Queue的第一筆資料] => 最先出去

稱為佇列 & 排隊 (一串資料)

ADT需要:

1. 建構 2. 解構 3. 是否為空 4. 新增 5. 移除 6. 擷取 → 只動最早.

Create an empty... / Destroy a... / Is empty? / Add a new item / Remove / Retrieve

應用: palindrome 迴文.

Stack reverses vs queue reverse

=> 兩者取的第一位不同 (一個頭一個尾)

檢查是否迴文.

Queue: a b c b d (front, back) Stack: d b c b a ← top

比較 front and top 是否相同

=> 都相同 => 迴文.

```
isPal (int str : string) : boolean
  aQueue . createQueue()
  aStack . createStack()
  for (the next character ch in str){
    aQueue . enqueue (ch)
    aStack . push (ch) }
  charequal = true;
```

```
While (! aQueue.isEmpty() && charEqual){
  aQueue . dequeue (front)
  aStack . pop (top)
  if (front == top){
    aQueue.
    aStack.
  }
  if (front != top) charequal = false; }
```

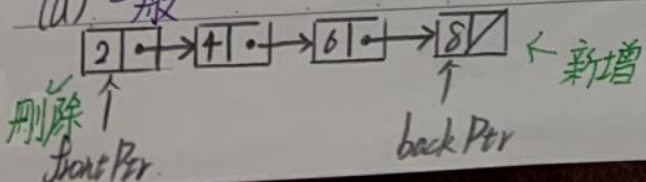
✳ (a) A linear linked list with two external references

=> front, back

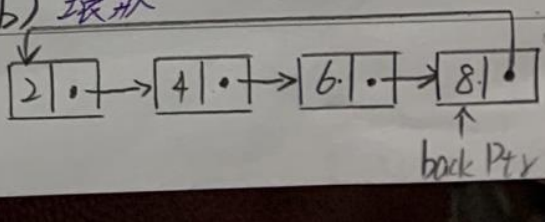
(b) A circular linked list with one external reference

=> back 環狀 => 只有後端

(a) 一般

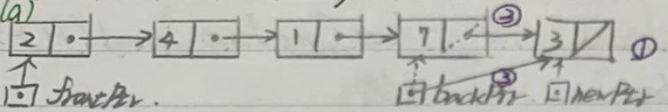


(b) 環狀



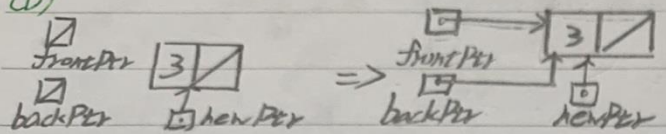
A Pointer-Based Implement.

新增 (a)



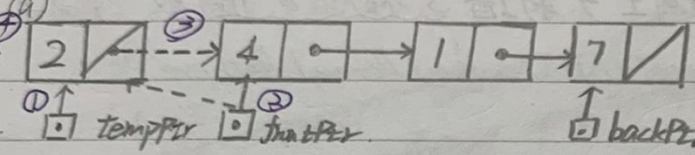
① newPtr → next = NULL;
② backPtr → next = newPtr;
③ backPtr = newPtr;

(b)



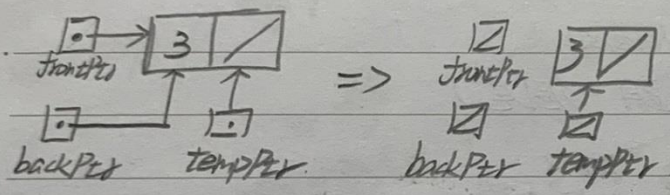
frontPtr = newPtr
backPtr = newPtr

移除 (a)



1 tempPtr = frontPtr;
2 frontPtr = frontPtr → next;
3 tempPtr → next = NULL;
4 delete tempPtr;

(b)



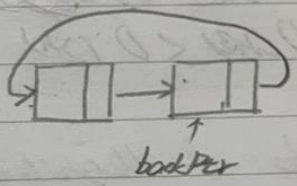
1 temp = frontPtr;
2 frontPtr = NULL;
3 back = NULL;
4 temp → next = NULL;
5 delete tempPtr;

```
void Queue :: dequeue () throw (QueueException) {
  if (isEmpty())
      throw QueueException ("QueueException: ...");
  else{
      Queue * tempPtr = frontPtr ①
      If (frontPtr == backPtr){
          frontPtr = NULL;
          backPtr = NULL; }
      else frontPtr = frontPtr -> next;
      temp -> next = NULL;
      delete tempPtr; }
}
```

Circular Queue

```
void Queue :: enqueue (const QueueItemType & newItem) {
  QueueNode *newPtr = new QueueNode;
  newPtr -> item = newItem;
  if (isEmpty()) newPtr -> next = newPtr
  else {
    newPtr -> next = backPtr -> next
    backPtr -> next = newPtr;
  }   backPtr = newPtr;
}

void Queue :: dequeue throw (QueueException) {
  if (isEmpty()) throw....
  else {
    QueueNode *tempPtr = backPtr -> next;
    if ( backPtr == backPtr -> next )
        backPtr = NULL;
    else backPtr -> next = tempPtr -> next;
    tempPtr -> next = NULL;
    delete tempPtr;
  }
}
```



3.

作業應用：Simulation

Arrival [抵達時間]	duration [時間長度]	Departure [完成時間]	waiting 等待時間
20	5	25	0 AWT = 10/4 = 2.5
22	4	29	3 MWT = 6 最大等待Time
23	2	31	6 MQL = 2 排隊長度
30	3	34	1 AQL = (1+2+1)

No.
Date: / /

Algorithm Efficiency 演算法效率

* Growth-rate function 成長函數

Definition of the order of an algorithm 大小階

⇒存在兩個常數 k and n_0 exist such that A requires no more than $k * f(n)$ time units to solve a problem of size $n \geq n_0$.

* 忽略低位階

$O(n^3+3n^2) \Rightarrow O(n^3)$

忽略常數

$O(5f(n)) = O(f(n))$

$O(f(n)) + O(g(n)) = O(f(n)+g(n))$

* Order of growth of some common function

$O(1) < O(\log_2 n) < O(n) < O(n \times \log_2 n) < O(n^2) < O(n^3) < O(2^n)$

* $\log_2 n = \log n$

Order-of-Magnitude Analysis and Big O Notation

Function	10	100	1000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
$n \times \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{30}	10^{18}
2^n	10^3	10^{30}	10^{301}	10^{3010}	10^{30103}	$10^{301,030}$

* Worst-case analysis 最多

Average-case analysis 平均

Best-case analysis 最少

No.
Date: / /

* Sequential search 循序搜尋 (有先排序有差)

Worst case : O(n)

Average case : O(n) => $\frac{(n+1)}{2}$

Best case : O(1)

* Binary Search of a sort array

* Repeatedly divide the array in half

* Determine which half could contain the item

Worst case : O(log2n)

Average case : O(log2n)

Best case : O(1)

* 內部排序

=> 資料量可被電腦記憶體容納。

若無法 => 外部排序

Stable Sort

insertion

merge

radix

(bubble)

Unstable Sort

selection

quick

heap

No.
Date: / /

[身高]

☼ bubble sort 泡泡排序 swap pass

每次執行n個資料，n-1次比較，n-1(比較好)

優：每比較完一次，都會有一個資料位置是正確

缺：一次一次換

[大隊接力]

☼ selection sort 選擇排序 swap

相似bubble sort，

優：每次都是比較到此間距最適合的位置

缺：個人覺得debug比較難，第一輪比完不一定正確

☼ Insertion Sort 插入排序 [撲克牌]

以第一位為基準，看後一項該插入在哪個位置，比較可由右往或相反，基準加一，越後面比較越多次

優：搬動次數少，缺 →

(公式)次數：

Bubble Sort：$n+2[n*(n-1)-n*(n-1)/2]+(n-1)=n^2+n-1 \Rightarrow O(n^2)$

Selection Sort：$n(n-1)/2 \Rightarrow O(n^2)$