6

Queues : 佇列 = 排隊

E.g.
① 先進來先服務 (汽車、GA表格)

ADT queue operations :

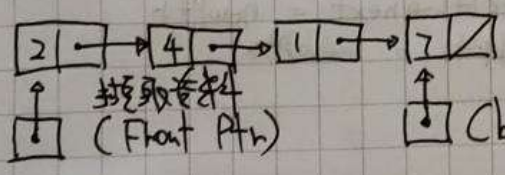
① creat an empty queue (建構)
② Dectory a queue (解構)
③ Determine whether a queue is empty (是否為空)
④ ADD a new item to the queue (新增)
⑤ Remove the item that was added earliest (移除)
⑥ Retrieve the item that was added earliest (擷取)

E.g.

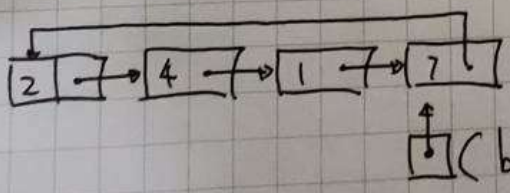
		Queue
a. createQueue()	(建構)	
a. enqueue (5)	(新增)	5
a. enqueue (2)	"	5 2
" " (7)	"	5 2 7
a. getFront (queueFront)	(擷取)	5 2 7 (q.F is 5)
a. dequeue (queueFront)	(移除)	2 7 (q.F is 5)
a. dequeue (queueFront)	"	7 (q.F is 2)

- A linear linked list with two external references (線性)
  - A reference to the front → delqueue 前端
  - A reference to the back → enqueue 後端
- A circular linked list with one external reference (環狀)
  - only a reference to the back 只有後端

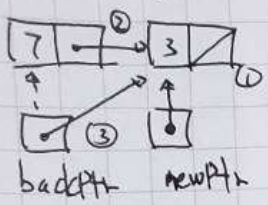
a)



b)

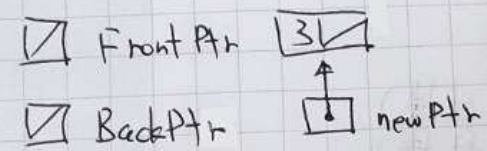


新增 enqueue

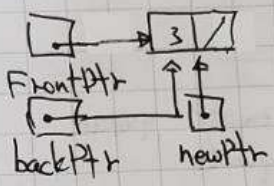


1. newPtr → next = Null;
2. backPtr → next = newPtr;
3. backPtr = newPtr;

(a) 一開始是空的

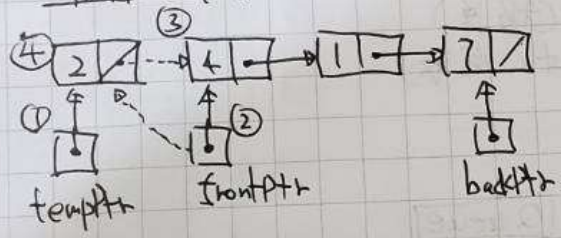


(b)



frontPtr = newPtr
backPtr = newPtr

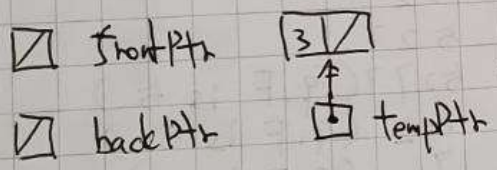
移除 dequeue



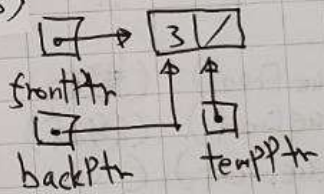
1. tempPtr = frontPtr;
2. frontPtr = frontPtr → next;
3. tempPtr → next = Null;
4. delet tempPtr;

queue 內之有一個 → 移除到空

a)



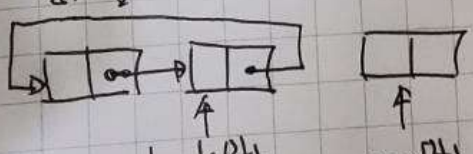
b)



1. tempPtr = frontPtr;
2. frontPtr = Null;
3. backPtr = Null;
4. tempPtr → next = Null;
5. delet tempPtr;

Circular Queue

① 新增



1. newPtr → next = backPtr → next
2. backPtr → next = newPtr
3. backPtr = newPtr

Simulation

20	5
22	4
23	2
30	3

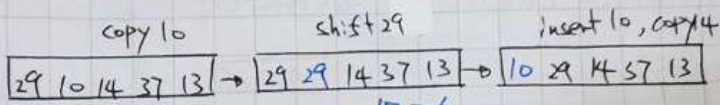
Time	Queue (front to back)	anEventList
0	(沒有排隊+服務) (empty)	A 20 5
20	20 5	(empty)
	20 5 (還沒有完成)	D 25 (預計在25min離開)
	(讀入下個客戶) 20 5	A 22 4 D 25 (下個客戶) → 排到後面
22	(Queue不是空的) → 排隊 20 5 22 4	D 25
	(讀入下個客戶) 20 5 22 4	A 23 2 D 25
23	(Queue前的第一個到25才完成) → 繼續排隊 20 5 22 4 23 2	D 25
	(讀入下個) 20 5 22 4 23 2	D 25 A 30 3
25	(20 5 完成了 進行 22 4 的任務) 22 4 23 2	A 30 3
	25+4	
	22 4 23 2	D 29 A 30 3

Events (input file)

Arrival	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

插入排序法 (Insertion Sort)

e.g. 摸扑克牌



n-1個回合

```
for ( i = 1 ; i < n ; i++ ){
    int loc = i , nextItem = A[i];
    for ( ; loc > 0 && (A[loc-1] > nextItem); --loc)
        A[loc] = A[loc-1];          // 逐一往後搬動
    A[loc] = nextItem;              // 直到 NextItem 的位置
} // for
```

希爾排序法 (shell sort)

```
for (int h = n/2 ; h > 0 ; h = h/2 ){          h → 間隔
    for (int unsorted = h ; unsorted < n ; unsorted++){
        int loc = unsorted;
        int nextItem = A[unsorted];
        for ( ; loc >= h && (A[loc-h] > nextItem); loc = loc-h)
            A[loc] = A[loc-h];
        A[loc] = nextItem;
    } // for
} // for
```

a)

20 80 40 25 60 10 15 // 有7比資料 → 7/2 = 3(間隔)

unsorted = h = 3 從25開始，往右走，且每一個數字都和左邊間隔h的排序

e.g. 60和80(3間隔) 10和40

20 60 40 25 80 10 15

20 60 10 25 80 40 15

到15的時候，往左間隔的有3次(20, 25, 15) 3個一起比較

15 60 10 20 80 40 25

※ 希爾排序法並不是 stable (穩定的)

資料排序的時候會產生互換動作，但是在遠距離互換的時候，沒有辦法分辨數值一樣的數在前面，誰在後面

7

Selection Sort (選擇排序法)

* 最佳情況下，相等就不做交換。

```
void selectionSort (int A[], int n) {
    for (last = n-1; last > 0; last--) { // 從最後 → 0
        int largest = indexOfLargest(A, last+1); // 找到最大的數字
        if (largest != last) // 不相同才交換
            swap(A[largest], A[last]);
    } // for
} // selectionSort

int indexOfLargest( int A[], int size) {
    int indexSoFar = 0;
    for (i=0; i < size; i++) // 從0開始歷遍
        if (A[indexSoFar] < A[index]) // 找到更大的數字
            indexSoFar = i; // 換成更大的數字的index
    return indexSoFar;
} // index Of Largest
```

Bubble Sort (氣泡排序法)

```
void bubbleSort (int A[], int n) {
    bool sorted = FALSE; // Flag
    for (pass = 1; pass < n && !sorted; pass++) {
        sorted = TRUE; // 先預測為已經完成
        for (int index = 0; index < n-pass; index++) {
            if (A[index] > A[index+1]) { // 只要發現需要排序的
                swap(A[index], A[index+1]);
                sorted = FALSE; // 尚未完成排序
            } // if
        } // for
```

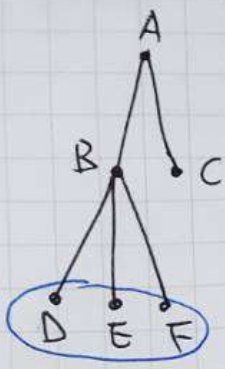
Mergesort (合併排序)

```
Void   mergeSort (DataType  Array[], int  first, int  last){
    mergeSort ( Array , first  , mid); // 以遞迴分組至剩最小
    mergeSort ( Array,  mid+1 , last);
    merge  (Array , first, mid, last); // 邊合併邊排序
}// mergeSort

Void   merge (DataType  Array[] , int first, int mid,  int last) {
    DataType   temp [MAX_SIZE];
    int first1    = first, last1   = mid ;
    int first2    = mid+1  , last2  = last;
    int  index  = first ;     // 暫存數組存到的位置
    for ( ;(first1 <= last1) && (first2 <=last2); index++) {
        if( Array[first1] < Array[first2]) {
            temp [index] = Array[ index];
            first1++ ;
        }//if
        else{
            temp[index] = Array[ first2];
            first2++
        }//else
    }// for
    for (; first1 <= last1 ; first1++, index++) // 可能還有剩下的
            temp [index]= Array [first1];
    for ( ; first2 <= last2 ; first2 , index++) //
            temp[index] = Array[first2];
    for (index = first ; index <= last ; index++) // 排序好的數組
            Array [index] = temp [index];   歸還
```

8

Trees



- Parent- child 親子關係
- Ancestor- descendent 祖孫關係
- subtree of a tree 子樹

葉節點 Leaf
兄弟節點 siblings

① A是B的父節點
② B是A的子節點
③ 葉節點：沒有子節點
④ 兄弟節點：同一個父節點
⑤ 祖先節點：最上層的節點

二元樹：

一個節點下最多只能有2個節點

★ 樹不能迴環

完全樹：

所有分叉都填滿

完整樹：

(1) 樹高-1 → 完全樹
(2) 最後一層都在左邊

平衡樹：

左右樹高的差距不超過1