Day4: 搜尋、排序、堆疊、佇列 07/11

C++ STL

C++ STL（標準範本庫）是一套功能強大的 C++ 範本類別，提供了通用的範本類別和函數

這些範本類和函數可以實現多種常用的演算法和資料結構，如 vector、set、list、queue、stack。

大理石在那裡(Where is the Marble? Uva 10474)

現有 N 個大理石，每個大理石上寫了一個非負整數。

首先把各數從小到大排序，然後回答 Q 個問題。

每個問題問是否有一個大理石寫著某個整數 x，

如果是，還要回答哪個大理石上寫著 x。排序後的大理石從左到右編號為 1～N。

範例輸入：	範例輸出：
4 1 2 3 5 1 5 5 2 1 3 3 3 1 2 3	CASE #1: 5 found at 4 CASE #2: 2 not found 3 found at 3

解析：

- 第一行 4 1 表示有 4 個數，1 個問題
- 第二行是大理石上的 4 個數
- 第三行是問題
- 輸出是 5 在排序後的第 3 塊大理石上找到。
- 這題思路很簡單，先進行排序，再查找提問的數。

```
#include <cstdio>
#include <algorithm>
using namespace std;
const int maxn=10000;
int main()
{
    int n,q,x,a[maxn],kase=0;
    while(scanf("%d%d",&n,&q)==2&&n)
    {
        printf("CASE# %d:\n",++kase);
        for(int i=0;i<n;i++)
            scanf("%d",&a[i]);
        sort(a,a+n);   //排序
        while(q--)
```

```
        {
            scanf("%d",&x);
            int p= lower_bound(a,a+n,x)-a;
        //在已排序陣列a中尋找x,lower_bound的作用是查找“大於或者等於x的第一個位置”
            if(a[p]==x)
               printf("%d found at %d\n",x,p+1);
            else printf("%d not found\n",x);
        }
    }
    return 0;
}
```

說明：

- 這題中使用了 algorithm 標頭檔中的 **sort** 和 **lower_bound** 功能
- sort 將元素從小到大（昇冪）排序，並且可以對任意物件進行排序。
- 如果希望用 sort 排序，這個類型需要定義“小於”運算子，或者在排序時傳入一個“小於”函數。排序物件可以存在普通陣列裡，也可以存在於 vector 中。
  - 陣列用法 ：**sort（a，a＋n）**
  - vector 用法：**sort（v.begin()，v.end())**
- **lower_bound** 的作用是查找“大於或者等於 x 的第一個位置”。

Linear Search / Sequential Search

```
#include < stdio.h >
/* Search for key in the List */
int seq_search(int key, int a[], int n)
{
    int i;
    for (i = 0; i < n; i++)
    {
        if(a[i] == key) return i + 1;
    }
    return 0;
}

int main()
{
    int i, n, key, pos, a[20];
    scanf("%d", &n);
    for (i = 0; i < n; i++)
        scanf("%d", &a[i]);
    scanf("%d", &key);
    pos = seq_search(key, n, a);
    if (pos == 0)
        printf("Not found\n");
    else
        printf("key found at %d\n", pos);
    return 0;
}
```

Binary Search 二分搜尋法

```
#include <stdio.h>
#include <stdlib.h>
int binarysearch(int[], int, int);
int main()
{
    int key, pos;
    int a[] = {3, 7, 14, 20, 23, 32, 41, 44, 56, 57, 73, 89, 93};
   scanf("%d", &key);
    // 呼叫函式進行搜尋
    pos= binarysearch(a, key, sizeof(a)/sizeof(int));
    if (pos < 0)
       printf("Not found %d\n", key);
   else
        printf("found %d at %d\n", key, pos + 1);
    return 0;
}
int binarysearch(int a[], int key, int n)
{
    int low = 0, high = n - 1;
    while (low <= high)
    {
        int mid = (low + high) / 2;
        if (a[mid] == key)
        {
            return mid;
        }
        else if ( key < a[mid]  )
        {
            high = mid - 1;
        }
        else if ( key > a[mid] )
        {
            low = mid + 1;
        }
    }
    return -1;
}
```

C++中的順序容器

#include **<vector>** //不定長陣列

- #include **<map>** //映射
- #include **<set>** //集合
- #include **<list>** //鏈表
- #include **<deque>** //*雙向佇列
- #include **<queue>** //佇列
- #include **<stack>** //堆疊

不定長陣列 vector

vector 是一個範本類(template)，所以需要用 vector<int> a

vector<double> b

這樣的方式來宣告一個 vector

vector 基本操作

```
vector<int> a;
```

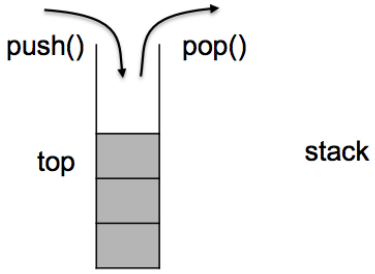
- a.size( );　　//讀取 a 的大小
- a.resize( );　　//改變 a 的大小
- a.push_back( );　　//向 a 尾部添加元素
- a.pop_back( );　　//刪除 a 的最後一個元素
- a.clear();　　//清空 a
- a.empty();　　//測試 a 是否為空

堆疊、佇列與優先佇列

STL 提供 3 種特殊的資料結構：堆疊、佇列與優先佇列。

堆疊(Stack)

- 堆疊(Stack): 符合「後進先出」(Last In First Out, LIFO)規則
- 可 PUSH 和 POP.



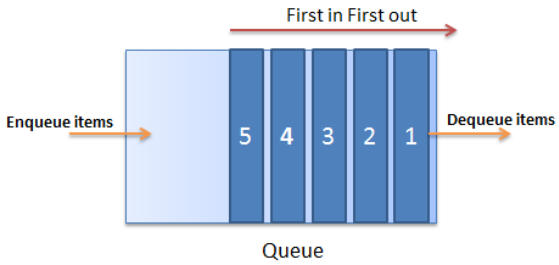

```
#include <stack>
stack<int> intStack;
```

- intStack.empty()：如果 stack 為空，則返回 true，否則返回 false；
- intStack.size()：返回 stack 中元素的個數
- intStack.pop()：刪除，但不返回 stack 頂元素
- intStack.top():返回,但不刪除 stack 頂元素
- intStack.push(item):放入新的 stack 頂元素

佇列 queue

- #include **<queue>**
- **queue<int>** s；
- s.**empty();**　//如果佇列為空，則傳回 true，否則傳回 false;
- s.**size();**　　//傳回佇列中元素的個數;
- s.**push(item);**　//在佇列尾放入一個新元素。
- s.**pop();**　　//刪除佇列首元素，但不傳回佇列首元素。

- s.**front()**;　　//傳回佇列首元素,但不刪除佇列首元素
- s.**back()**;　　//傳回佇列尾元素,但不刪除佇列尾元素。



優先佇列 priority_queue

- #include **<queue>**
- priority_queue**<int>** s；
- s.**empty()**;//如果佇列為空，則傳回 true，否則傳回 false;
- s.**size()**;//傳回佇列中元素的個數;
- s.**pop()**;//刪除佇列首元素，但不傳回。在 priority_queue 中，佇列首元素代表優先順序最高的元素;
- s.**push(item)**;//在佇列尾放入一個新元素。對於 priority_queue,將根據優先順序排序
- s.**top()**;//傳回優先順序最高的元素但不刪除

Ugly Numbers Uva 136

Ugly Numbers 是指不能被 2，3，5 以外的其他質數整除的數。
把 Ugly Numbers 從小到大排列起來，結果如下：1,2,3,4,5,6,8,9,10,12,15,…
求第 1500 個 Ugly Number

Note:

1. 除了 1 之外，其餘所有的 Ugly Numbers 都是由最基本的 2，3，5 這三個 Ugly Numbers 相乘或者自乘得到的。也就是說，兩個 Ugly Numbers 相乘的積是 Ugly Numbers 的充分必要條件；
2. 將 2，3，5 作為種子數位，不斷生成 Ugly Numbers 。等到生成足夠多的 Ugly Numbers 後，就可以得到第 1500 個 Ugly Number 了；
3. 以最開始為例，我們取出隊首元素 2，依次和種子數字 2，3，5 生成三個新的 Ugly Number：4，6，10，然後檢查是否已經出現過；再取出隊首元素 3，重複這個步驟；
4. 要注意的一點是，在生成的過程中可能會產生重複的數，要注意排除；

```
#include <iostream>
#include <vector>
#include <queue>
#include <set>
using namespace std;
typedef long long LL;
```

```
//考慮到第 1500 個 ugly number 的計算過程中可能已經超過 int 範圍，需要用到 long long 類型
const int coeff[3]={2,3,5};
int main()
{
    priority_queue<LL,vector<LL>,greater<LL>> pq;
    set<LL> s;  //s 用於存儲所有 ugly number
    pq.push(1);
    s.insert(1);  //單獨把第一個 ugly number 1 分別加入到優先佇列和 s 中
    for(int i=1;;i++)
    {
        LL x=pq.top();   //將隊首元素（最小的元素）取出
        pq.pop();  //將隊首元素（剛取出的最小元素）從優先佇列中刪除
        for(int j=0;j<3;j++)
        {
            LL x2=x*coeff[j];  //生成三個新的 ugly number
            if(!s.count(x2))
            {
                s.insert(x2);
                pq.push(x2);
            }
        }
        if(i==1500)
        {
            cout<<"The 1500'th ugly number is "<<x<<".\n";
            break;
        }
    }
    return 0;
}
```

排序

排序的基本認識

1. 所謂排序是將一群資料按照某一個特定規則重新排列，使其具有遞增或遞減的次序關係。
2. 按照特定規則，用以排序的依據，我們稱為鍵（Key），它所含的值就稱為鍵值。
3. 排序依資料量的多寡有內部排序（Internal Sort）和外部排序（External Sort）之分。資料量小可以全部載入記憶體來進行者稱為內部排序，大部分排序屬於此類；資料量大無法全部一次載入記憶體滿必須借磁帶、磁碟等輔助記憶體者稱為外部排序。外部排序在排序過程中要考慮 I/O Buffer 的問題。
4. 另外，依排序的過程可以區分為直接移動與邏輯移動兩種。直接移動是直接移動資料的位置；而邏輯移動則是改變資料的指標位置。
5. 幾個相同的鍵值排序後，相同的鍵值仍然保持原來的次序，稱為穩定排序（Stable Sort）。

一、氣泡排序法(Bubble Sort)

1. 比較陣列中相鄰兩元素之鍵值，若兩元素之次序不對，則將兩元素值交換。
2. 步驟：
   - 相鄰之兩資料項 X(i)與 X(i-1)互相比較。
   - 若次序不對則將兩資料項對調，直到不產生對調為止。
   - 重複以上動作，直到 N-1 次或互換動作停止。
3. 氣泡排序是屬於 Stable Sort。
4. 最壞的狀況是資料以相反次序排列，共需 N(N-1)/2 次比較；最好的狀況是資料已排序好或所有鍵值都相同時，共需(n-1)次比較。

二、選擇排序法(Selection Sort)

1. 將所有資料項中找一個最小(或最大)鍵值之資料項置於第一個位置。然後再由剩餘的資料項中，找出次小（或次大）的，依此類推。
2. 步驟：
   ・找出第 i 個至第 N 個鍵值中最小者，並將之與第 i 個鍵值交換。(第一次 i=1)
   ・重覆以上動作，直到 i=n-1 為止。
3. 選擇排序是屬於 Unstable Sort。
4. 選擇排序是需要 n-1 次對調與 n(n-1)/2 次比較。
5. 當資料項是以相反次序排列時，此方法最快。

三、插入排序法(Insertion Sort)

1. 將該鍵值插入到其前面所有鍵值當中，第一個大於本身鍵值之前，若沒有則插入在最後面。
2. 步驟：
   甲、 將該第 i 個鍵值插入到其前面所有鍵值當中，第一個大於本身鍵值之前，若沒有則置於最後面。
   乙、 重覆以上動作，直到 i=n-1 為止。

✧ 插入排序是屬於 Stable Sort。

✧ 所需額外空間很少。

四、快速排序法(Quick Sort)

1. 快速排序法是一種分而治之（Divide and Conquer）的排序法，它的觀念是將待排序的 n 個鍵值分成左右兩半，左半邊之鍵值小於第一個鍵值，而右半邊則大於或等於第一個鍵值。
2. 步驟：
   (1) 令 K=第一個鍵值（最左邊之鍵值）。
   (2) 由左向右找出一個鍵值 Ki，滿足 Ki>K。
   (3) 由右向左找出一個鍵值 Kj，滿足 Kj<K。
   (4) 若 i<j，則將 Ki 與 Kj 交換，然後重覆(2)至(4)。
   (5) 若 I>=j，則將 K 與 Kj 交換，並以 j 為基準分割成左右兩半，然後分別針對左右兩半進行步驟（一）至（五），直到左半邊鍵值=右半邊鍵值為止。

✧ 快速排序是屬於 Unstable Sort。

✧ 快速排序具有遞迴本質。

✧ 最壞狀況是當資料項的次序是完全相反。

✧ 快速排序將檔案一分為二，先取其中一部分排序，另一部分則將其位置指標暫存於堆疊中。

五、合併排序法(Merge Sort)

1. 合併排序是將兩個或兩個以上已排序之資料集，合併成一個大的已分類的資料集。
2. 步驟：
   (1) 將 n 個長度為 1 的鍵值成對地合併成 n/2 個長度為 2 的鍵值組。
   (2) 將 n/2 個長度為 2 的鍵值組成對地合併成 n/4 個長度為 4 的鍵值組。
   (3) 將鍵值組成對地合併，直到合併成一組長度為 n 的鍵值組為止。

✧ 合併排序是屬於 Stable Sort。

✧ 亦是一種分而治之（Divide and Conquer）的排序法。

六、堆積排序法(Heap Sort)

1. 每一個父節點資料必大於或等於它兩個子節點資料。
2. 步驟：
   (1) 產生完整二元樹。
   (2) 產生堆積樹。
   (3) 輸出樹根（並以最後樹葉取代）。

(4) 回步驟 1。

✧ 堆積排序是屬於 Unstable Sort。

✧ 輸出為小→大，則過程中之 output 置於 stack；輸出為大→小，則 outpput 置於 queue。

✧ heap 是一個 complete binary tree，每一個節點之值均大於或等於其子節點之值，樹根是 heap 中之最大值。

✧ heap 的特性是 root 為最大，因此 priority queue 若用 heap 來處理效率會較佳。

✧ heap 結構可用 array 來製作成 priority queue。

七、基數排序法(Radix Sort)

1. 基數排序法又稱為多鍵排序（Multi-Key Sort），或箱子排序法（Bucket Sort）。
2. 基數排序法依其比較之方向分為最有效鍵優先和最無效鍵優先兩種。
3. 最有效鍵優先（Most Significant Digit First, MSD）是從最左邊的位數開始比較，是採用分配、排序、收集等三個步驟進行。其中排序是使用插入排序法。
4. 最無效鍵優先（Least Significant Digit First, LSD）是從最右邊的位數開始，只須採用分配和收集兩個步驟。
5. 例如：利用 LSD 方式排列正整數鍵值。

   ■ 首先按個位數分配到相同的箱子中，收集後得到的順序為：

   231, 871, 63, 585, 165, 66, 58

   ■ 按拾位數分配到正確的箱子中，收集後得到的順序為：

   231, 58, 63, 165, 66, 871, 585

   ■ 按百位數分配到正確的箱子中，收集後得到的順序為：

   58, 63, 66, 165, 231, 585, 871

➢ 基數排序是屬於 Stable Sort。

➢ 基數排序所需的額外空間很大。且其中 LSD 無法排序負數。

八、樹排序法(Tree Sort)

1. 首先將鍵值建造成二元搜尋樹，即樹中的每一節點均滿足(1)大於或等於左子樹根，(2)小於或等於右子樹根，因此：

   (1) 令第一個鍵值為樹根。

   (2) 第二個到第 n 個鍵值依序建到樹中，從樹根開始比，若比樹根小，則放到左子樹，否則放到右子樹。
2. 以中序法追蹤即為所得。
3. 樹排序法是屬於 Stable Sort。

十、外部排序法(External Sort)

1. 資料量太大，無法全部載入記憶體。
2. 將資料切割成多個區段，各區段利用內部排序法排序結果亦存於輔助記憶體。
3. 利用合併排序法合併各區段。

4. 優點：能排序較大的檔案。

5. 缺點：內部排序中，資料鍵值在記憶體內的比較、搬移或交換位置是屬於電子動作，速度較快。但是外部排序 I/O 需要花費不少時間，所以效率很差。

效益評估

排序法	最壞情況所花時間	平均所花時間	屬於穩定排序	是否交換位置	所須額外記憶體空間	備　註
氣泡	$O(N^2)$	$O(N^2)$	Stable	是	O(1)	N 小較好
選擇	$O(N^2)$	$O(N^2)$	Unstable	是	O(1)	N 小較好，部分已排序較好
插入	$O(N^2)$	$O(N^2)$	Stable	是	O(1)	大部已排序較好
謝耳	$O(N^S)$	$O(N(\log_2 N)^2)$	Stable	是	O(1)	N 小較好 1<S<2
快速	$O(N^2)$	$O(N\log_2 N)$	Unstable	是	O(logN) ~ O(N)	很好
堆積	$O(N\log_2 N)$	$O(N\log_2 N)$	Unstable	是	O(1)	很好
合併	$O(N\log_2 N)$	$O(N\log_2 N)$	Stable	否	O(N)	常用於外部排序
基數	$O(N\log_r B)$	$O(N\log_r B)$ ~ O(N)	Stable	是(MSD) 否(LSD)	O(N*S)	B：箱子個數 r：基數
樹	$O(N^2)$	$O(N\log_2 N)$	Stable	否	O(1)	好