為 Water Jugs Problem 鋪路

請先參考 jugs.pptx 投影片的說明，認識一下我們想要解決的問題。在開始解 Water Jugs Problem 之前，我們必須先介紹一些 C++ Standard Library 裡面的幾個常用的 containers 的基本用法。

範例 `map_example.cpp`

這個範例和之前的作業 WordCount 的問題相同，要計算輸入的英文字出現的次數。
適合用來當作 `map` 的練習。

`map<string, int> WordFreq;` 產生了一個物件 `WordFreq` ，是由 `string` 對應到 `int` 的一種資料儲存方式。尖括號裡面第一個參數 `string` 代表 key，第二個參數 `int` 是 value，我們可以用 key 來查詢對應的 value 是多少。 `map` 可以使用跟陣列一樣的 `[]` 運算符號。在 C++ 稱作 map (因為是做從 key 到 value 的 mapping)，不過由於作用很像字典，在其他語言也稱作 dictionary。

用 `while (cin >> str)` 讀取鍵盤輸入的資料，暫存到字串變數 `str` 裡面，然後用 `str` 當作 key，來查詢 `WordFreq` 這個 map 裡面是否已經有儲存過相同資料。如果有，就用 `WordFreq[str]++;` 讓次數加一，如果沒出現過，就設定 `WordFreq[str] = 1;` 。
如何判斷 map 裡面是否已經有相同資料？第一種方式是用 `find()` 函數。

```
    iterator find (const key_type& k);
```

如果找不到相同 key 的資料， `find()` 會傳回一個指向 map 結尾的 iterator (先想成是 pointer)，所以我們只需要拿來和 `WordFreq.cend()` 或 `WordFreq.end()` 比較，就能判斷是否找到了對應的資料。

```
        if (found == WordFreq.cend()) {
            // a new word, initialize its count as 1
            WordFreq[str] = 1;
        } else {
            // an existing word; increase its frequency by 1
            WordFreq[str]++;
        }
```

判斷是否已在 map 裡面，除了用 `find()` 函數之外，第二種方式是直接用 `count()` ，用法如下

```
        if (WordFreq.count(str)==0) {
```

```
            // a new word, initialize its count as 1
            WordFreq[str] = 1;
        } else {
            // an existing word; increase its frequency by 1
            WordFreq[str]++;
        }
```

map 的 `count()` member function，只會傳回兩種值，`0` 代表要找的 key 不在 map 裡，`1` 則表示已經在 map 裡。

當資料都已經用迴圈讀取完畢，都放入 map 並且累計次數之後，剩下的事情就只需要把 map 的內容，依序取出並顯示到螢幕上。由於 map 本來就會自己依照 key 的大小排序，所以如果 key 是 `string`，就會用英文字母的順序來排。我們用 `for` 迴圈，把 map 裡面每一筆資料取出，每一筆資料可以看成 `pair` 的形式，`.first` 就是 key，`.second` 則是 value。

底下是完整的程式碼。

```
#include <iostream>
#include <algorithm>
#include <map>
using namespace std;
int main()
{
    // create a map (a dictionary) that maps a string to an integer
    // map: key->value where
    // key: string, value: int
    map<string, int> WordFreq;

    string str;
    while (cin >> str) {
        // use the idiom of finding an element in a container
        auto found = WordFreq.find(str);
        if (found == WordFreq.cend()) {
            // a new word, initialize its count as 1
            WordFreq[str] = 1;
        } else {
            // an existing word; increase its frequency by 1
            WordFreq[str]++;
        }
    }
```

```
    // iterate through the data in the map
    // the data are sorted by the keys
    // .first is the key and .second is the value
    for (auto it=WordFreq.cbegin(); it!=WordFreq.cend(); ++it) {
        cout << (*it).first << ": " << (*it).second << '\n';
    }
}
```

最後面的迴圈，可以改成 range-based for loop，會更簡潔。

```
    for (auto w : WordFreq) {
        cout << w.first << ": " << w.second << '\n';
    }
```

範例 `vector.example.cpp`

請看底下的程式碼。先產生物件 `vector<int> vec;` 。然後搭配 `cin` 和 `while` 迴圈讀取資料，用 `push_back()` 函數來將資料放入 `vec` 後面。
資料讀完之後，接來可以用至少四種不同迴圈寫法，把 `vector` 裡面存放的資料取出來。

- 第一種是用迴圈跑過所有的 index，然後用陣列的寫法， `vec[i]` 取出對應編號的元素。
- 第二種則是用 range-based for loop，在這個例子中，因為只是單純的要把資料取出來顯示，這應該是最合適的寫法，不需要擔心是否會超出 `vector` 範圍。
- 第三種是用 `for_each` ，必須要先 `#include <algorithm>` 才能使用。通常 `for_each` 會搭配 lambda 函數使用，可以做顯示之外，更多複雜的操作。因為這個例子只是要顯示資料，我們需要的 lambda 是

  ```
  [](const int v){cout << v << " "; }
  ```

  暫時可以把 `[]` 想成是一個沒名字的函數，接收一個參數 `v` ，函數內做的事情是把 `v` 印出來。 `for_each` 會拿這個函數，依序套用在 vector 裡面的每一個元素 (用 `vec.cbegin()` 和 `vec.cend()` 來指定要處理的範圍)，所以 vector 裡面的每個元素，會輪流扮演參數 `v` ，然後被 `cout` 顯示到螢幕上。

- 第四種，把 iterator 當指標來用。

這個範例的第二段，主要是介紹如何用 `sort` `random_shuffle` `any_of` `none_of` `all_of` 這些函數來處理 vector 資料。 `sort` 和 `random_shuffle` 的用法應該很單純，看了範例就會知道。至於 `any_of` `none_of` `all_of` ，則都是要搭配額外的函數或是 lambda 函數，當作第三個參數傳入 (把函數當作參數傳入 `any_of` 或 `none_of` 或 `all_of`)。當作參數傳入的函數，會套用在 vector 的每個元素上面，依據條件判斷的結果，傳回 `true` 或 `false` ，然後如果是

`any_of` ，只要任一元素傳回 `true` ，整個 vector 對於 `any_of` 得到的結果就會是 `true` ，如果是 `none_of` ，則是每個元素都是 `false` ，整個 vecotr 對於 `none_of` 才會是 `true` 。至於 `all_of` 則是每個元素都必須是 `true` 整個 vector 對於 `all_of` 才會是 `true` 。

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;
int main()
{
    vector<int> vec;
    int i;

    // We can grow a vector incrementally using push_back().
    // To stop cin, type Ctrl-Z+Enter at the beginning of a line.
    cout << "Enter some integers ";
    cout << "and type Ctrl-Z + Return at the beginning of a newline.\n";
    while (cin >> i) {
        vec.push_back(i);
    }

    // There are several way to iterate through a vector.
    // They can be treated as idioms.
    // 1. Access the elements by indexes and []
    for (unsigned int i=0; i<vec.size(); i++) {
        cout << vec[i] << ' ';
    }
    cout << endl;

    // 2. Use C++11 range-based for loops.
    for (auto v : vec) {
        cout << v << ' ';
    }
    cout << endl;

    // 3. Use for_each (need to include <algorithm>) and C++11 lambda functions
    for_each(vec.cbegin(), vec.cend(),
             [](const int v){cout << v << " "; }
             );
```

```
    cout << endl;

    // 4. Use an iterator
    for (auto it=vec.cbegin(); it!=vec.cend(); ++it) {
        cout << *it << ' ';
    }
    cout << endl;

    // Other types of vectors
    vector<string> heroes{"Ironman", "Batman", "Superman", "Spiderman", "Thor"};
    for (auto s : heroes)
        cout << s << ' ';
    cout << endl;

    cout << "Sorted: \n";
    sort(heroes.begin(), heroes.end());
    for (auto s : heroes)
        cout << s << ' ';
    cout << endl;

    cout << "Shuffle: \n";
    random_shuffle(heroes.begin(), heroes.end());
    for (auto s : heroes)
        cout << s << ' ';
    cout << endl;

    // functions in <algorithm>
    // all_of, none_of, any_of
    if ( all_of(heroes.cbegin(), heroes.cend(), [](string s){ return s.length()>3;
}) )
        cout << "All names have more than three letters.\n";
    else
        cout << "false\n";
}
```

範例 `set_example.cpp`

- 把資料放入 set，可以用 initializer list 的方式，在產生物件的時候，用 copy constructor 把資料複製到 set 裡面。

- 也可以用 `insert()` 函數，把新的資料放入 set 裡面。
- 在 set 裡面找資料，也是用 `find()` 函數。

範例的後半段，示範如何用 `next_permutation` 來產生不同的排列，而且把結果放入 `set<vector<int>>` ，set 裡面的每個元素都是一個由 int 構成的 vector，。

範例的最後，則是示範如何用 `bitset` 來窮舉 powerset。
`bitset` 可以把整數轉成二進位表示法，我們可以透過查詢每個位元是 `0` 或 `1` ，來決定是否要把對應到該位元的資料挑選出來。

```
#include <iostream>
#include <set>
#include <algorithm>
#include <bitset>
using namespace std;  // otherwise we need to write std::set

int main()
{
    set<int> S{1, 2, 3, 4};

    // insert a new element into a set
    for (auto c : S)
        cout << c << ' ';
    cout << '\n';
    cout << "insert 5: ";
    S.insert(5);
    for (auto c : S)
        cout << c << ' ';
    cout << '\n';

    // insert an existing element into a set
    S.insert(1);
    cout << "insert 1: ";
    for (auto c : S)
        cout << c << ' ';
    cout << '\n';

    // We use the following idiom to find an element in a set.
    auto x = S.find(3); // x is an iterator
    if (x != S.cend()) {
        cout << "Found: " << *x << '\n';
```

```
    }


    // Generate all permutations and store them in a set.
    vector<int> v{1,2,3};
    set<vector<int>> T;
    do {
        T.insert(v);
    } while ( next_permutation(v.begin(), v.end()) );

    for (auto c : T)
        cout << c[0] << ' ' << c[1] << ' ' << c[2] << '\n';
    cout << '\n';

    // using bitset to generate power sets
    vector<string> vs{"RED","GREEN","BLUE"};
    for (int i=0; i<8; ++i) {
        bitset<3> b(i);
        for (int j=0; j<3; ++j) {
            if (b[j]==1) {
                cout << vs[j] << ' ';
            }
        }
        cout << '\n';
    }


}
```

進入正題: 寫程式解出 Water Jugs Problem

需要一個基本的資料結構，用來描述每個水桶在某個時刻的狀態，`using State = vector<int>;`。
例如 {3, 0} 代表第一個水桶裡面有三加侖的水，另一個則是空的。

接下來我們要定義 `class Pouring`，用來描述 Water Jugs Problem 的倒水過程。
`class Pouring` 包含幾個 private members:

```
    vector<int> _capacities; // {3, 5}
```

```
set<State> _explored;
set<list<State>> _paths;
set<list<State>> _solutions;
```

- `_capacities` 用來記錄每個水桶的容量上限。
- `_explored` 用來標記已經出現過的狀態，避免重複。
- `_paths` 則是一堆 `list<State>` 的集合， `list<State>` 構成了一條路徑(一種解法、一種倒水順序)，我們把全部的路徑都放在 `_paths` 裡面，解題的過程中，可以取出任意一條路徑，然後試著往下走。
- `_solutions` 跟 `_paths` 一樣，不過只儲存能夠達成目標的解法。

`class Pouring` 的 constructor 只需要把 `_capacities` 設定好。

按照題目定義，我們可以做三種操作： `Empty` 、 `Fill` 、 `Pour` ，把水桶的水倒光、把水桶的水裝滿、把某個水桶的水倒到另一個水桶。我們必須實做出這三項操作。

```
State Empty(State s, int jug_no);
State Fill(State s, int jug_no);
State Pour(State s, int from, int to);
```

再來是 `set<State> extend(State s)` 函數，傳入的參數是某個狀態，然後依據狀態，用上述的三種操作，把狀態改變，然後把產生的新狀態，通通存入 `set<State>` 裡面。

最關鍵的函數是 `void solve(int target, int steps)` 。其中 `target` 是我們想要達到的目標，某個桶子裡面有指定的水量。 `steps` 則是希望在多少步驟之內解出答案。
主要的想法如下 (從 jugs.pptx 節錄出來)

1. For each path in the set of candidate paths
   1 Extract the last state of that path and add it into the set of explored states
   2 Extend the last state to get a set of next states that have not been visited yet
   3 For each of the new next states, append it to the end of the current path and therefore obtain a new path

2. If the new state is the target, we are done;

3. Otherwise , add the new path into the set of candidate paths.

對照程式碼會更清楚上述的步驟如何實現。

完整的程式碼如下:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <set>
```

```
#include <list>
#include <iterator>
#include <string>
#include <sstream>
using namespace std;
using State = vector<int>;
class Pouring
{
private:
    vector<int> _capacities; // {3, 5}
    set<State> _explored;
    set<list<State>> _paths;
    set<list<State>> _solutions;

public:
    Pouring(vector<int> cp): _capacities{cp} { }
    State Empty(State s, int jug_no)
    {
        s[jug_no] = 0;
        return s;
    }
    State Fill(State s, int jug_no)
    {
        s[jug_no] = _capacities[jug_no];
        return s;
    }
    State Pour(State s, int from, int to)
    {
        State t = s;
        int diff = _capacities[to]-s[to];
        if (diff < s[from]) {
            t[to] = _capacities[to];
            t[from] = s[from]-diff;
        } else {
            t[from] = 0;
            t[to] = s[to] + s[from];
        }
        return t;
    }
    set<State> extend(State s)
```

```
{
    set<State> SS;
    for (int i=0; i<_capacities.size(); ++i) {
        SS.insert(Empty(s, i));
        SS.insert(Fill(s, i));
        for (int j=0; j<_capacities.size(); ++j) {
            if (i!=j)
                SS.insert(Pour(s, i, j));
        }
    }
    return SS;
}

void show_state(State s)
{
    for (auto i : s)
        cout << i << ", " ;
    cout << "->";
}
bool found(State s, int target)
{
    for (auto t : s) {
        if (t==target) return true;
    }
    return false;
    // return any_of(s.cbegin(), s.cend(), [=](int v){ return target==v;});
}
void solve(int target, int steps)
{
    list<State> initialPath;
    initialPath.push_back(State(_capacities.size()));
    _paths.insert(initialPath);

    while (steps > 0) {
        set<list<State>> newPaths;
        set<list<State>> oldPaths;

        for (auto p : _paths) {
            _explored.insert(p.back());
            auto nextStates = extend(p.back());
```

```
                for (auto s : nextStates) {
                    if (found(s, target)) {
                        auto np = p;
                        np.push_back(s);
                        _solutions.insert(np);
                    } else {
                    auto search = _explored.find(s);
                    if (search == _explored.cend()) {
                        auto np = p;
                        np.push_back(s);
                        newPaths.insert(np);
                    }
                    }
                }
                oldPaths.insert(p);
            }

            for (auto p : oldPaths) {
                _paths.erase(p);
            }
            for (auto p : newPaths) {
                _paths.insert(p);
            }
            --steps;
        }

    }
    void show_solutions()
    {
        for (auto path : _solutions) {
            for (auto state : path) {
                show_state(state);
            }
            cout << "\n";
        }
    }
};

int main()
{
```

```
    vector<int> jugs = {3, 5};
    Pouring problem(jugs);
    problem.solve(4, 6);
    problem.show_solutions();
}
```