程式設計（105-2）
作業六

作業設計：孔令傑
國立臺灣大學資訊管理學系

繳交作業時，請至 PDOGS（`http://pdogs.ntu.im/judge/`）為第一、二題上傳一個 PDF 檔，再為第三題與第四題各上傳一份 C++ 原始碼（以複製貼上原始碼的方式上傳）。第四題是 bonus 加分題。每位學生都要上傳自己寫的解答。不接受紙本繳交；不接受遲交。請以英文或中文作答。

這份作業的截止時間是 ***2017 年 4 月 10 日凌晨一點***。在你開始前，請閱讀課本的第 7.1–7.9 和 7.11 節[1]。為這份作業設計測試資料並且提供解答的助教是李昱賢（Rick Lee）。

第一題

（40 分；每小題 10 分）請回憶一下作業五第二題，昱賢提出的以遞迴方式求解「C 幾取幾」的演算法。那個方法有個好處，就是比較能避免溢位（overflow），但缺點是需要比較長的執行時間。傑夫仔細地思考了一下這個遞迴解法，並且觀察了敬傑寫的範例程式碼（在作業五的解答裡），發現之所以需要比較長的執行時間，是因為「有些問題被重複地求解了」。比如說要算 $C(10,5)$ 時，我們需要算 $C(9,5)$ 和 $C(9,4)$，而要算 $C(9,5)$ 需要算 $C(8,5)$ 和 $C(8,4)$，要算 $C(9,4)$ 則需要算 $C(8,4)$ 和 $C(8,3)$。我們可以發現 $C(8,4)$ 被重複求解了，而在敬傑的程式碼裡面，是貨真價實地要花兩份時間來做這一份工作。整個求解 $C(10,5)$ 的過程當然就有很多時間被浪費在求解這些一樣的題目上了。

有鑑於此，傑夫想到了兩個加速（不要浪費時間）的作法：

(a) 傑夫打算在使用者輸入 n 和 m，並且要求程式計算 $C(n,m)$ 之後，宣告一個 $n \times m$ 的二維矩陣 A，將每個元素都初始化為 -1。傑夫預計要在 A_{ij} 儲存 $C(i,j)$ 的值，如此一來，每當想要求解 $C(i,j)$ 之前，就先檢查一下 A_{ij} 是否為 -1，如果是 -1 就去求解，並用求解後的結果取代 -1，否則就直接回傳 A_{ij} 即可。

由於 n 和 m 是使用者輸入的值，因此傑夫打算宣告一個二維的動態陣列 `com`，並且修改遞迴函數 `combiRec` 為

```
int combiRec(int n, int m, int** com)
{
  if(n < m)
    return -1;
  else if(n == m)
    return 1;
  else if(m == 1)
    return n;
  else
  {
    if(com[n - 1][m - 1] == -1)
```

[1]課本是 Deitel and Deitel 著的 *C++ How to Program: Late Objects Version* 第七版。

```
    {
      int res = combiRec(n - 1, m, com) + combiRec(n - 1, m - 1, com);
      com[n - 1][m - 1] = res;
      return res;
    }
    else
      return com[n - 1][m - 1];
  }
}
```

請用你自己的話描述上面這段程式碼，並說明它如何實現傑夫的設計。

(b) 接著 main function 也需要做相對應的改寫，以便宣告並且初始化 `com`。傑夫寫好了 main function 的架構如下：

```
int main()
{
  int n = 0, m = 0;
  cin >> n >> m;

  // declare an n by m dynamic array
  // and initialize all elements in it to -1

  cout << combiRec(n, m, com);
  return 0;
}
```

請幫傑夫完成註解那部份的宣告和初始化。請注意 `int**` 和二維靜態陣列是不一樣的，把 `com` 宣告成一個靜態二維陣列丟進 `combiRec` 是會產生 syntax error 的！完成之後，請試著輸入 $n = 35$ 和 $m = 15$，雖然結果是錯的（因為溢位），但是確實會比之前快多了。

(c) 我們在上面使用的方法，說到底就是「解過的題目不要再解一次」，但本質上還是一個 top-down 的設計：「我要解一個大問題，就拆成一些小問題去分別解決」。在同一種思維模式下，其實有另一種 bottom-up 的設計常常是更好的：「我從最小的問題開始往上解，以便每當我需要解一個比較大的問題時，它底下的小問題都已經被解好了，我只要查一下表即可」。這類型演算法被通稱為**動態程式規劃**（dynamic programming）。

我們用 Fibonacci 數列來當例子。還記得在之前上課時，我們曾寫過程式根據給定的 n，找第 n 個 Fibonacci 數，而我們利用那個例子說明了遞迴的作法可能會浪費很多時間。當時的實作基本上是這樣：

```
int fib(int n)
{
  if(n <= 2)
    return 1;
  else
```

```
    return fib(n - 1) + fib(n - 2);
}
```

如果我們改採用上面介紹的 top-down 作法，可以把程式改寫成：

```
int fibRec(int n, int* fibValue)
{
  if(n <= 2)
    return 1;
  else
  {
    if(fibValue[n - 1] == -1)
    {
      int res = fibRec(n - 1, fibValue) + fibRec(n - 2, fibValue);
      fibValue[n - 1] = res;
      return res;
    }
    else
      return fibValue[n - 1];
  }
}
```

其中 `fibValue` 是儲存前 n 個 Fibonacci 數的一維動態陣列。如果我們改採用 bottom-up 的動態程式規劃，可以把程式改寫成：

```
int fibDP(int n, int* fibValue)
{
  fibValue[0] = fibValue[1] = 1;
  for(int i = 2; i < n; i++)
    fibValue[i] = fibValue[i - 1] + fibValue[i - 2];
  return fibValue[n - 1];
}
```

可以看到 `fibDP` 的 header 和 `fibRec` 的 header 是一模一樣的，亦即 `fibValue` 依然是用來儲存前 n 個 Fibonacci 數的一維動態陣列。然而現在我們是從最小的子問題（$k = 1$ 和 2）開始往比較大的子問題（$k = 3, 4, \ldots$）去解，直到我們解出原本要解的問題（$k = n$）為止。

現在，請改寫傑夫在 (a) 小題寫的 `combiRec` 函數，將之以動態程式規劃的方式實作出來。

(d) 請比較 (a) 和 (c) 小題的兩種實作方式，並且用自己的話說明兩者各有什麼「在時間複雜度上的」好處跟壞處。

第二題

（20 分；每小題 10 分）請回答以下兩個彼此之間沒有任何關聯的問題：

(a) 請針對本次作業的第四題所指定的演算法，撰寫一個 pseudocode。請盡量讓你的 pseudocode 詳細且精確，接近可以被直接翻譯成程式碼（但還不是程式碼）的樣子，像作業三第六頁的第二個 pseudocode 那樣。你的 pseudocode 愈詳細且精確，就會得到愈高分。

注意：第四題是加分題，但這題是基本題。

(b) 請考慮第六講投影片第 61 頁的程式碼：

```
int main()
{
  int r = 3;
  int** array = new int*[r];
  for(int i = 0; i < r; i++)
  {
    array[i] = new int[i + 1];
    for(int j = 0; j <= i; j++)
      array[i][j] = j + 1;
  }
  print(array, r); // later
  // some delete statements
  return 0;
}
```

佩蓉在 `// some delete statements` 那邊嘗試寫了一句 `delete` 敘述

```
  delete [] array;
```

但是昱賢跟她說這樣還是會發生 memory leak，因為那三條一維陣列的空間都沒有被釋放。他說，得要搭配一個迴圈去釋放那三條一維陣列，才是這裡需要的完整釋放動態記憶體空間的作法。請幫佩蓉寫出完整且正確的程式碼，來釋放所有被動態配置的記憶體空間。你的答案可能包含佩蓉原本寫的，也可能不包含；若你覺得應該包含，請把佩蓉原本寫的那句包含在你的答案裡。

第三題

（40 分）在本題中，我們將重新實作一次作業五的第三題（根據歷史交易記錄與消費者的購買品項對其做推薦）。之所以要重新實作一次，是因為實務上一個零售店通常有成千上萬個品項，但消費者一次通常只買十幾個品項，因此如果總品項數為 n、總交易次數為 m，而我們用一個 $n \times m$ 的靜態二維陣列來存交易記錄，就會有非常非常多的陣列元素都是存 0[2]。仔細想想，如果是 Amazon.com 這種有幾千萬個品項的「店」，那二維靜態陣列這種資料結構也實在浪費空間到一個人神共憤的程度了。除此之外，使用靜態陣列意味著貴店不能增加品項，能容許的交易次數也有其上限，這些都會讓你的程式很不具彈性。因此在這一題，我們要來練習用動態二維陣列儲存歷史交易資料。當然資料結構更改之後，後續的運算也都得更著做修正[3]。

[2]或許有些同學在寫作業五時，就已經不是用靜態二維陣列當資料結構了。有鑑於這樣程式會非常難寫（特別是不能使用上課沒教過的方法），讓我們假設大家都是用靜態二維陣列吧。

[3]要是真的有幾千萬個品項和幾百億個交易，我們這門課介紹的任何資料結構和演算法都不足以針對推薦問題做良好的運算。你肯定需要更好的資料結構和更好的演算法，而這可能是你得要去修更進階的課的動機吧。

在我們即將實作的陣列中，我們依然會有 m 列，一列代表一次交易，但現在我們在一列中將只儲存該次交易中被購買的品項的編號。因為兩次交易中購買的品項數未必相同，所以我們的每一列都會是一個動態陣列，長度恰好為該次交易被購買的品項數。換言之，我們的「表格」將會「各列長度不一」，而且「表格」的列數是可以持續增加的（不過在這一題中，我們不會要求你做這件事）。

請修改你的（或助教提供的）程式碼，來改變你的資料結構以及相對應的運算。

題外話：如果你以前沒有太多寫程式的經驗，你說不定會發現與其修改你的程式碼，還不如砍掉重練算了。這可能是因為你原本的程式模組化不夠、變數名稱不清不楚，註解寫得不夠多，或者任何其他原因。如果你經營一家公司，你肯定希望貴公司在更新資訊系統時，不是全部砍掉重練，而是在既有系統上做修改和增補；如果貴公司是軟體公司，或你自己是軟體工程師，專門生產軟體系統賣給別的公司或人，那就更不用說了。能寫出「對」的程式之後，下一步就是寫出「好」的程式了。一個程式要好需要滿足很多面向的要求，而「容易被看懂」和「容易被修改和擴充」就是一個重要的面向。不論從什麼時候開始都好，試著寫出好的程式吧！

輸入輸出格式

本題的輸入輸出格式和作業五的第三題一模一樣。

你上傳的原始碼裡應該包含什麼

你的.cpp 原始碼檔案裡面應該包含讀取測試資料、做運算，以及輸出答案的 C++ 程式碼。當然，你應該寫適當的註解。針對這個題目，你***不可以***使用上課沒有教過的方法。

本題有兩個特殊要求：

1. 你必須使用二維動態陣列來儲存交易資料，而且兩個維度（列跟欄）都必須是動態的。
2. 你必須寫一個函數

```
void setTransactions(int** trans, int* itemCnt, int m);
```

其中 `trans` 是儲存歷史交易資料的二維動態陣列、`itemCnt` 是儲存每筆交易中各購買了幾個品項的一維動態陣列、`m` 是由測試資料讀入的歷史交易資料筆數。你的 main function 必須呼叫此函數一次，在此函數中讀入測試資料中的歷史交易資料，並將相關資訊記錄在 `trans` 和 `itemCnt` 中，以便後續程式做處理和運算。

評分原則

- 這一題的其中 40 分會根據程式運算的正確性給分。PDOGS 會編譯並執行你的程式、輸入測試資料，並檢查輸出的答案的正確性。前 30 分由 15 筆測試資料判定分數，一筆測試資料佔 2 分；後 10 分由 5「組」測試資料判定分數，每一組裡面有若干筆測試資料，全對的話才能得到 2 分。
- 這一題的其中 20 分基本上是送分。助教會打開你的原始檔，檢查你的程式碼是否滿足本題的兩個特殊要求。如果有，就給你 20 分，就算你在 PDOGS 上拿到零分。但如果沒有，不但這 20 分會完全拿不到，助教也有權力在情節嚴重時扣你在 PDOGS 上得到的分數。最壞的情況下（例如你直接上傳在作業五得到滿分的程式碼），你全部的分數都會被扣光，亦即這一題會得 0 分。

- **特殊事項：**在作業七中，我們會開啟一個新的挑戰：讓同學們互相檢視彼此的本題程式碼，並且就可讀性、易維護性、模組化程度、排版等面向寫評語和給評分（當然一切都是匿名的）。細節會在作業七中描述。該任務在作業七中會佔 20 分，其中 10 分取決於檢視你的程式碼的同學給你的分數（必要時助教會出來主持公道，請不用緊張），另外 10 分取決於你對同學的程式碼的評語和評分的合理性和建設性。因此，雖然有點殘酷，但你若在作業六中沒有寫這一題，那作業七自然沒有人能檢視你的程式碼，你也就得要損失那 10 分了。

  其實你也不需要煩惱這麼多；好好寫這一題就對了！

第四題（bonus）

（40 分）如果你在 lab exam 1 沒有寫出第四題（決定補貨策略那一題），請不要難過，因為只有不到五分之一的同學有寫出來。現在機會來了，在本題中，你將會重新面對他並且解決它。請注意這一個加分題佔 40 分，是有史以來最多分的加分題。如果你考試真的考得不太理想，就多寫點加分題吧[4]！

當然，我們不能讓你去下載助教寫的答案再上傳就賺到 40 分，所以我們把題目做了一點變化。原本題目是說各物流中心的容量是無上限的，也就是說就算你要讓一個物流中心對所有的零售店補貨也沒問題。實務上這當然是不太可能，因此在本題中，我們用 K_j 表示物流中心 j 的容量上限，亦即物流中心 j 最多只能放 K_j 個商品。整個問題就變成設定 x_{ij}（物流中心 j 對零售店 i 的補貨數量）來求解

$$\begin{aligned}
\max \quad & \sum_{i=1}^{m}\sum_{j=1}^{n}(p-cd_{ij})x_{ij} \\
\text{s.t.} \quad & \sum_{j=1}^{n} x_{ij} \le D_i \quad \forall i = 1, ..., m \\
& \sum_{i=1}^{m} x_{ij} \le K_j \quad \forall j = 1, ..., n \\
& x_{ij} \ge 0 \quad \forall i = 1, ..., m, j = 1, ..., n。
\end{aligned}$$

而其中新增的第二條限制式就表現出物流中心的容量限制。

在這個情況下，要求出能最大化總利潤的補貨計畫，雖然做得到，但就不是這麼容易了。因此，我們要請你實作下面這個簡單的演算法。你的演算法將會執行許多回合，在每個回合中你會挑出一對物流中心跟零售店做補貨。每個回合中你將依序執行三個步驟：

1. 首先，找出還有未滿足需求的零售店集合 T，對每一個還有未滿足需求的零售店 $i \in T$ 去找出「對該零售店補貨能帶來正的銷貨毛利的尚有剩餘容量」的物流中心集合 S_i，並在這個集合 S_i 中挑出離該零售店最近的物流中心 j_i^*。更精確地說，j_i^* 滿足

   $$d_{i,j_i^*} \le d_{ij} \quad \forall j \in S_i。$$

   如果沒有任何物流中心能對它帶來正的銷貨毛利，則 S_i 是空集合，j_i^* 自然就不存在；如果有多個物流中心都能帶來正的銷貨毛利且有剩餘容量，且他們都是離零售店 i 最近的，就選剩餘容量最大的；再平手，就挑物流中心編號最小的。

2. 現在對每個還有未滿足需求的零售店 i，你都有找到一個物流中心 j_i^* 了。請對每一對 (i, j_i^*) 計算他們之間的距離 d_{i,j^*i}，並且在這些距離中挑出最短的距離，稱呼那個離自己的物流中心 j_i^* 最近

[4]作業是得幾分就算幾分，超過一百分也照算，所以最後算學期成績時，是有機會用作業成績去補其他項目的成績的。

的零售店 i 為 i^*。更精確地說，i^* 滿足

$$d_{i^*,j^*_{i^*}} \le d_{i,j^*_i} \quad \forall i \in T。$$

如果有多個零售店 i 離其物流中心 j^*_i 都同樣是最短距離，就挑未滿足需求量最大的；再平手，就挑零售店編號最小的。

3. 挑出 i^* 以及其物流中心 $j^*_{i^*}$ 後，就由 $j^*_{i^*}$ 補貨給 i^*，而補貨數量是 $j^*_{i^*}$ 的剩餘容量與 i^* 的未滿足需求中比較小的那個量。

完成這三個步驟之後，請進入下一回合再重新執行這三個步驟，直到某個回合挑不出 i^* 為止。此時已經沒有任何一個有未滿足需求的零售店，能找到一個有剩餘容量的物流中心對其補貨會帶來正的銷貨毛利了。理論上此時我們還可以再多做一些補貨，來滿足盡量多的需求。為了題目簡單起見，在這題中我們就到此為止即可。換句話說，請不要做任何不能增加利潤的補貨[5]。

輸入輸出格式

系統會提供許多筆測試資料，每筆測試資料裝在一個檔案裡；前四行和 lab exam 1 的第四題一樣，第五行是新增的。在每個檔案中，第一列存放四個整數 n、m、p 和 c。第二列存放 $2n$ 個整數 u_1、v_1、u_2、v_2 直到 u_n、v_n，表示第 i 個物流中心在位置 (u_i, v_i) 上。第三列存放 $2m$ 個整數 x_1、y_1、x_2、y_2 直到 x_n、y_n，表示第 i 個零售店在位置 (x_i, y_i) 上。第四列存放 m 個整數 D_1、D_2 直到 D_m，表示第 i 個零售店的需求量是 D_i。第五列存放 n 個整數 K_1、K_2 直到 K_n，表示第 j 個物流中心的流量上限是 K_j。每一列中的任兩個數字之間都用一個空白鍵隔開。已知 $1 \le n \le 10$、$1 \le m \le 1000$、$1 \le p \le 500$、$1 \le c \le 10$、$0 \le x_i \le 200$、$0 \le y_i \le 200$、$0 \le u_i \le 200$、$0 \le v_i \le 200$、$1 \le D_i \le 100$，以及 $1 \le K_j \le 100000$。原本題目中就有的參數的範圍都跟 lab exam 1 的第四題中給的範圍一樣。

請根據題目的描述以及給定的演算法，找出能最大化總銷貨毛利的補貨計畫，並且印出兩個整數，分別代表在我們找到的補貨計畫下能賺到的總銷貨毛利以及未被滿足的需求總量。舉例來說，如果輸入是

```
2 6 5 1
2 2 4 5
1 1 1 2 2 5 3 1 7 2 9 4
3 4 5 6 7 8
16 3
```

表示物流中心 1 的容量是 16、物流中心 2 的容量是 3。若我們執行給定的演算法，則每一回合我們會依序做下面的運算與補貨：

- 回合 1：$T = \{1,2,3,4,5,6\}$；$j^*_1 = 1$、$j^*_2 = 1$、$j^*_3 = 2$、$j^*_4 = 1$、j^*_5 跟 j^*_6 不存在；$i^* = 2$ 且 $d_{i^*,j^*_{i^*}} = 1$。我們從物流中心 1 對零售店 2 補貨 4 個商品，物流中心 1 的剩餘容量變成 12。
- 回合 2：$T = \{1,3,4,5,6\}$；$j^*_1 = 1$、$j^*_3 = 2$、$j^*_4 = 1$、j^*_5 跟 j^*_6 不存在；$i^* = 4$ 且 $d_{i^*,j^*_{i^*}} = 2$。我們從物流中心 1 對零售店 4 補貨 6 個商品，物流中心 1 的剩餘容量變成 6。請注意零售店 1、3、4 各自距離物流中心 j^*_1、j^*_3、j^*_4 都是 2 公里，因此我們挑未滿足需求最大的零售店 4。

[5]如果你已經對於被指定演算法感到厭煩了，請不要生氣；在期中專題我們將會讓你（強迫你）設計你自己的演算法。

- 回合 3：$T = \{1, 3, 5, 6\}$；$j_1^* = 1$、$j_3^* = 2$、j_5^* 跟 j_6^* 不存在；$i^* = 3$ 且 $d_{i^*, j_{i^*}^*} = 2$。我們從物流中心 2 對零售店 3 補貨 3 個商品，物流中心 2 的剩餘容量變成 0、零售店 3 的未滿足需求變成 2。
- 回合 4：$T = \{1, 3, 5, 6\}$；$j_1^* = 1$、$j_3^* = 1$、j_5^* 跟 j_6^* 不存在；$i^* = 1$ 且 $d_{i^*, j_{i^*}^*} = 2$。我們從物流中心 1 對零售店 1 補貨 3 個商品，物流中心 1 的剩餘容量變成 3。
- 回合 5：$T = \{3, 5, 6\}$；$j_3^* = 1$、j_5^* 跟 j_6^* 不存在；$i^* = 3$ 且 $d_{i^*, j_{i^*}^*} = 3$。我們從物流中心 1 對零售店 3 補貨 2 個商品，物流中心 1 的剩餘容量變成 1。
- 回合 6：$T = \{5, 6\}$，但 j_5^* 跟 j_6^* 都不存在，因此不存在 i^*，演算法結束。

綜合以上，我們找到的補貨方式是讓物流中心 1 對零售店 1、2、3、4 各補 3、4、2、6 個商品，而物流中心 2 對零售店 3 補 3 個商品。如此則賺得的總利潤是 $3(5-2)+4(5-1)+[3(5-2)+2(5-3)]+6(5-2) = 56$ 元，而零售店 5 和 6 的需求都完全沒有被滿足。因此，輸出應該是

```
56 15
```

請注意我們可以在不損害利潤的情況下，讓物流中心 1 對零售店 5 補貨 1 個商品，去滿足更多需求，但在本演算法下我們不這麼做。因此，輸出應該是 `56 15`，而不是 `56 14`。

你上傳的原始碼裡應該包含什麼

你的.cpp 原始碼檔案裡面應該包含讀取測試資料、做運算，以及輸出答案的 C++ 程式碼。當然，你應該寫適當的註解。針對這個題目，你***可以***使用任何方法。

評分原則

這一題的 40 分都根據程式運算的正確性給分。PDOGS 會編譯並執行你的程式、輸入測試資料，並檢查輸出的答案的正確性。前 30 分由 15 筆測試資料判定分數，一筆測試資料佔 2 分；後 10 分由 5「組」測試資料判定分數，每一組裡面有若干筆測試資料，全對的話才能得到 2 分。