CPP0902 Coding Assignment

訂貨及存缺貨問題

電子檔 2010/05/23 09:00 前上傳至 moodle
書面檔 2010/05/23 09:15 前於課堂繳交

運用在工業管理的範圍

模擬系統與分析，生管存貨。

相關 C++課程內容

讀檔，函式，陣列、動態記憶體的運用與配置。

問題背景與概述

假設有一 3C 量販店之門市販售自家品牌電腦，其每日的顧客需求量(D)為一個隨機整數（譬如昨天無顧客購買、今天有 3 台電腦的需求、明天有 2 台電腦需求等等），且根據歷史銷售記錄可歸納出每日的需求符合 Poisson 機率分配(如式(1))

$$p(x) = \frac{e^{-\lambda}\lambda^{x}}{x!} \qquad (1)$$

其中 e=2.718281828459….為自然對數函數之底數（或稱 Euler's number），x 為發生的需求量（假設為非負整數），而 λ 則表示平均每天發生之需求量（通常為題目給定）。在給定 λ 值之後，藉由式(1)代入不同之需求 x 值，即可得該需求值在每天發生之機率 p(x)(如表 1)。

表 1　$\lambda=4$，$K=9$ 之每日需求量與其對應之機率分配(假設百分比皆四捨五入取整數位)

每日需求 x (台)	0	1	2	3	4	5	6	7	8	9
機率 $p(x)$	2%	7%	15%	20%	20%	16%	10%	6%	3%	1%

表 1 中，$\lambda=4$ 代表每天平均有 4 個需求。不同之 λ 值會導致各需求的發生機率隨之不同。假設已知需求量為[0,9]區間之整數，則最大可能發生的需求量值 $K=9$。將 $\lambda=4$ 及 $x=0,1,...,9$ 代入式(1)所得到的 $p(x)$ 分別代表需求為 0,1,…,9 之發生機率。理論上 $\sum_{x=0}^{K} p(x)=100\%$，因此 K 的值必須足夠大（譬如 $K\geq 2\lambda+1$），否則會導致 $\sum_{x=0}^{K} p(x)<100\%$。為避免發生此不合理情況，本題目假設式(1)僅適用於計算 $x=0,1,...,K-1$ 之機率，而 $x=K$ 之機率則以 $100\%-\sum_{x=0}^{K-1} p(x)$ 反推算之。

站在增加營收的立場上，商家會希望盡可能滿足顧客需求，亦即希望每天皆可將店中的電腦完全清空賣出。若每日在打烊盤點時尚有部分電腦在庫（此即為存貨），則商家必須為每台在庫之電腦付出一筆「單位存貨成本(H)」；反之，若盤點時發現當天有顧客無法當場取貨（此即為缺貨），則每筆無法完成的買賣每延期交貨一日必須付出一筆「單位缺貨成本(S)」。存貨與缺貨其實是一體兩面，可用以下式(2)計算之：

```
I[t] = I[t-1] + O[t-L] – D[t], t=1,...,T        (2)
```

其中 t 為時間（以日為單位），I[t]、O[t]、D[t]分別代表第 t 日的存貨（或缺貨，為當日打烊前盤點結果）、訂貨量（為當日開店前決定）、預測需求量。當 I[t]>0 時，代表第 t 天有存貨；當 I[t]<0 時，代表第 t 天有缺貨；當 I[t]=0 時，代表第 t 天無存（缺）貨。而 L 代表每次訂

貨時送出訂單至收到貨品中間所耗費的「前置時間」；舉例來說，每筆門市之訂貨自第 t 日早上送出訂單至其上游工廠製造後，會於第 t+L 日早上接收到該批貨物。

有關 I[],O[],D[]，本作業假設如下：

- I[0]為給定，而 I[1],...,I[T]必須依式(2)推導而得；
- O[-L],...,O[0]皆為 0，而 O[1],...,O[T]必須依本作業規劃的四種訂價機制（兩種訂貨量及折扣、兩種訂貨時機）來設定；
- D[0]=0，D[T+1],...,D[T+L] 皆為 0，而 D[1],...,D[T]由給定的需求預測趨勢依電腦亂數產生而得。
- 每次訂貨需耗費一筆訂貨成本(C)，訂貨量的多寡及時機皆會影響整體營收，
- 在訂貨量的數量及折扣上有以下兩種機制（假設目前為第 t 日, t=1,…,T）：

[訂量1].O[t] = Q_1，即訂購 Q_1 單位，每單位貨品之製造成本為 M_1

[訂量2].O[t] = Q_2 ($>Q_1$)，即訂購 Q_2 單位，每單位貨品之製造成本為 M_2($<M_1$)

- 在訂貨的時機方面，本作業假設第 t 日開店前（t=1,…,T）有以下兩種機制：

[時機1].檢驗 I[t-1]再扣除第 t,…,t+L-1 日（不含第 t+L 日）產生的總需求量（即 I[t-1]-D[t]-D[t+1]-…-D[t+L-1]）是否小於「安全存貨(A)」。若是，則訂貨 O[t] = Q_1 或 Q_2 單位；反之，O[t] = 0。!!注意!! 此機制必須於每日早上執行一次。

[時機2].檢驗 I[t-1]是否小於再訂購點(R)。若是，則訂貨 O[t] = Q_1 或 Q_2；反之，O[t] = 0。

!!注意!!以上兩種機制一旦決定於第 t 日訂貨的話，則在第 t+1,…,t+L 日皆不可再訂貨。

其中，安全存貨是為了應付突發的需求，在系統中設定隨時應有的庫存量；以上假設之兩種訂貨時機及存貨機制之皆源自實務上生產管理中為減少存缺貨而常見的營運手法。

相關術語更詳細的解釋說明如下：

- 每日之顧客需求(Demand，**D[ ]**)：
  每一天的電腦需求量，在此題我們將其設為機率分佈(公式(1))已知的隨機整數。
- 單位製造成本(Manufacturing Cost，**M**)：本題設為 M_1 或 M_2
  生產一台電腦的總花費。
  E.g. 假設訂購 4 台電腦，總共將花費 $4M_1$ 或 $4M_2$ 元製造之。
- 單位缺貨成本(Shortage Cost，**S**)：
  在電腦已賣光無存貨的情形下，每產生一台電腦的需求隨即損失 S 元。
  E.g. 假設昨天已賣光電腦，而新訂的 4 台電腦後天才會到，則今明兩天將無電腦可賣。倘若今天及明天各有 2 台電腦需求，則因兩天缺貨，商家將付出 4S+2S=6S 元的代價。
- 單位存貨成本(Holding Cost，**H**)：
  倉庫每存放一台電腦（過夜）會造成 H 元的成本。
  E.g. 假設昨天在打烊後發現尚存 5 台電腦在庫，則這些（過夜）存貨將造成 5H 元成本。
- 單筆訂貨成本(Ordering Cost，**C**)：
  預估目前存貨可能無法滿足未來的需求時，將會向工廠訂貨。本題假設每天最多僅能訂一次（即一筆）貨，每次訂貨會產生 C 元的成本。
- 訂貨量(Ordering Quantity，**Q**)：本題設為 Q_1 或 Q_2

- 前置時間(Lead Time，**L**)：

  營運者於當日早上向工廠下訂單後，至營運者收到從工廠運來的電腦前，會有一段工廠的處理時間，其中包含製造時間、運送時間…等，總共花費的時間為 L 單位時間。

  E.g. 假設營運者預估無法供應未來產生的需求，於是今天向工廠下訂單，預計 2 天後（即於後天）才收到工廠送來的電腦，則整個前置時間為 2 天。

- 安全庫存(Safety Stock，**A**)：

  為了能降低缺貨的可能性，除了滿足事先預測可能產生的需求外，在許可範圍內為因應突然出現、所料未及的偶發需求，而多準備的存貨量 A 台。

  E.g. 假設預測未來三天的需求皆為 20 台電腦，營運者避免市場波動性太大而產生缺貨，故在倉庫額外多放置 3 台電腦，則這 3 台電腦就是安全庫存量。

- 再訂購點(Reorder Point，**R**)：

  從訂貨到收到貨物需要一段時間，避免在這段時間內因需求波動過大造成缺貨，故設置一「存貨水準」為再訂購點，也就是當存貨量低於該存貨水準（亦即，再訂購點）時，營運者將會向工廠發出訂單訂貨。

  E.g. 假設前置時間為 2 天，但市場波動性太大，使得營運者無法利用兩天前下訂單的策略，故設定當倉庫存貨低於 2 台電腦時，則向工廠下訂單，則 2 台為再訂購點。

作業目的及假設

本作業簡化現實中商家常面臨的「訂貨、存（或缺）貨」問題，以電腦程式測試模擬四種訂存貨策略對於不同的顧客需求趨勢所造成的營收差別。

假設總共欲模擬 T 日（即 t = 1,…,T），顧客的需求情況為已知的趨勢（在現實中可將此視為「需求預測」的結果，如表 1 所示），給定期初的存缺貨情況（I[0]），假設 O[0]=D[0]=0，試算營運者擬定訂存貨的數量及時機各兩種（總共四種）策略所導致的 O[t]、I[t]及其引發的成本，並將最佳策略（即花費最少）的結果列印出來。

程式要求及作法

本作業大概可切割成以下三個 PART：

PART1 讀輸入檔：以一函式

```
void initialize_p(int *T, int *I, int *K, int *l, double **p, int *M1,
                  int *M2, int *S, int *H, int *C, int *Q1, int *Q2, int *L,
                  int *A, int *R);
```

或

```
void initialize_r(int &T, int &I, int &K, int &l, double *&p, int &M1,
                  int &M2, int &S, int &H, int &C, int &Q1, int &Q2, int &L,
                  int &A, int &R);
```

實作之

本問題相關資料，包括 n、K、l，及諸如 M（包括 M_1, M_2）、S、H、C、Q（包括 Q_1, Q_2）、L、A、R 等參數必須存至一個名為「HW6_input.txt」的文字檔。其格式定義如下：

```
T   10          ← T = 10：總共模擬 10 日
i   2           ← I[0] = 2：期初之存缺貨量
K   9           ← K = 9：每日可能的需求值介於 0~9 之間
l   4           ← λ = 4：Poisson distribution parameter
```

```
M   500    350     ← M1 = 500, M2 = 350：兩種不同的單位製造成本
S   400         ← S = 400
H   100         ← H = 100
C   250         ← C = 250
Q   6   12      ← Q1 = 6, Q2 = 12
L   2           ← L = 2
A   3           ← A = 3
R   2           ← R = 2
```

程式中必須用 switch 判讀各列開頭字母以讀取其對應之參數值，並將之存入程式所定義之變數。在得知 l(即λ)與 K 值之後，必須先替 p[ ]陣列 allocate memory ，再回傳至 main()。

本部分困難之處：如何用 pointer 或 reference 讓函式可以一次回傳多個值(包括 array)？

如何開檔讀檔？

PART2 以隨機亂數產生對應的顧客需求量：以一函式

```
void GetProbability(int l, int K , double **p)
```

或 `void GetProbability(int l, int K , double *&p)` 依式(1)計算 p[]

(亦即，輸入 l,K 之後，本函式將依以下說明計算出 p[x],x=0,1,…,K)

其中，K 表示所有可能的需求量種類(以表 1 為例，K=9)；p[ ]記錄各種需求量對應之機率，其計算方式乃依式(1)計算 x=0,1,…,K-1 之機率 p[x] x=0,…,K-1；而 p[K]之機率再以 $100\%-\sum_{x=0}^{K-1} p[x]$ 反推算之(以表 1 為例，p[0]=2%,…, p[8]=3%, p[9]= (100-2-7-15-20-20-16-10-6-3)%=1%)。

計算出各需求值 x 之機率 p[x]後，必須使用以下函式來產生每日之隨機顧客需求量 D[t]:

```
int GetDemand(int K, double **p)
```

或 `int GetDemand(int K, double *&p)` 實作之

(亦即，輸入 K 及 p 之後，本函式將回傳一個[0,K]區間的隨機顧客需求量給 D[t],t=1,…T)

以下說明 GetDemand 函式之原理：因為 $\sum_{x=0}^{K} p(x)=100\%$，我們可由 p[x]來計算累積機率 $cp[Z]=\sum_{x=0}^{Z} p(x)$，Z=0,1,…,K。以 cp[0],cp[1],…,cp[K-1]當成分隔點，可將[0,1]區間分成 K+1 段。接著，假設我們產生一個值介於 0 至 1 之間的隨機實數，即可藉由判斷該隨機實數座落於這 K+1 段中的某段(假設為[cp[k'],cp[k'+1])那一段)，來反推其所對應的需求值為 k'。以表 1 之 p[x]為例，我們可將[0,1]區間分成 K+1=10 段如下：[0,0.02)、[0.02,0.09)、[0.09,0.24) 、[0.24,0.44) 、[0.44,0.64) 、[0.64,0.80) 、[0.80,0.90) 、[0.90,0.96) 、[0.96,0.99) 、[0.99,1.00)。倘若所產生的亂數值為 0.08，則回傳 1；同理，若所產生的亂數值為 0.70、0.91、0.29，則應分別回傳 5、7、3 來當隨機需求值（詳見圖 1.）。

藉由上述方式，我們可在主程式中以一個 for 迴圈(t=1,2,…,T)，藉由系統時間當 random seed 來一一產生 T 個介於 0 至 1 之間的隨機實數，並將之代入 GetDemand 函式以得到其所對應的顧客需求量，存入 D[t], t=1,…,T 中（D[0]先設為 0，D[T+1],…,D[T+L]亦設為 0）。

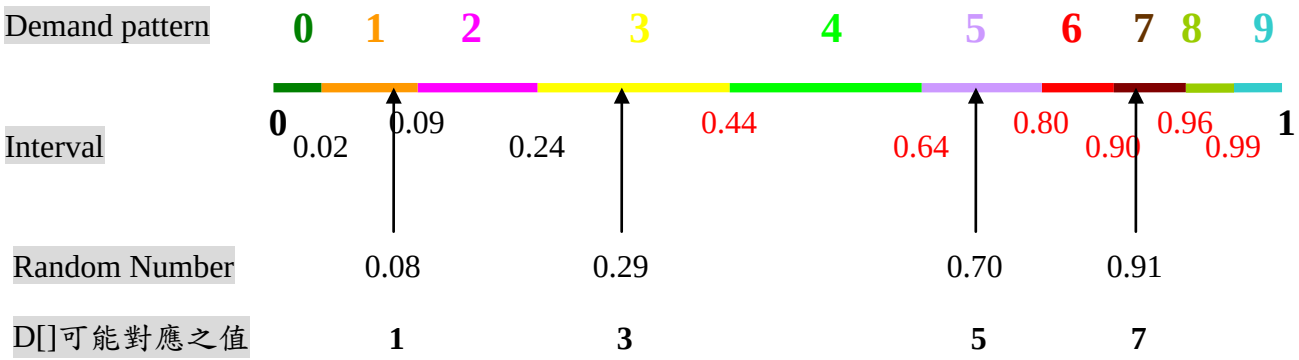

圖 1　根據隨機亂數以產生需求量

!!注意!! 雖然以上我們用 D[T+1],…,D[T+L]設為 0 來說明時間超過 T 的需求不用處理，其實 D[]這個 array 只要宣告其大小為 T+1 即可（即 t=0,1,…,T），對於超過 T+1 的部分一律以程式將之歸零即可。同理，在第一頁中提及的 O[-L],...,O[-1]亦可用類似想手法將之視為 0，連設置都不用，不必煩惱如何設定 index 為負數的 array。

PART3 計算四種機制(Plc[s],s=0,…,3)中各機制之各天存缺貨量 I[t]、訂貨量 O[t]、日花費 DC[t]，及其總花費：

本部分主要包括以下兩個子步驟：

PART3.1. 設定 4 種機制之相關參數及 allocate memory。

PART3.2. 針對每種機制，依其運作規則計算其天存缺貨量 I[t]、訂貨量 O[t]、日花費 DC[t]，及其總花費 TCost。

PART3.1 設定 4 種機制之相關參數及 allocate memory

首先，必須在程式開頭宣告一個名為「Policy」的 struct，其定義如下：

```
typedef struct {
   int M, Q;     //分別代表 M1 或 M2，Q1 或 Q2
   int *I, *O;   //分別代表 I[ ],O[ ]
   int *DC;      //代表日花費 DC[ ]
   int TCost;    //代表總花費
} Policy;
```

在 main()中先宣告 Policy Plc[4]，代表 Plc[0],…,Plc[3]等四種不同的訂存貨機制：

Plc[0]：其 $M=M_1$, $Q=Q_1$; 使用安全存貨量來決定訂貨時機（詳見第二頁）

Plc[1]：其 $M=M_1$, $Q=Q_1$; 使用再訂購點來決定訂貨時機（詳見第二頁）

Plc[2]：其 $M=M_2$, $Q=Q_2$; 使用安全存貨量來決定訂貨時機（詳見第二頁）

Plc[3]：其 $M=M_2$, $Q=Q_2$; 使用再訂購點來決定訂貨時機（詳見第二頁）

之後，必須針對各種機制之 I[ ],O[ ], DC[] allocate memory（這些 array 的元素個別有 T+1 個，t=0,1,…,T）。

PART3.2 計算各機制每天之存缺貨量 I[t]、訂貨量 O[t]、日花費 DC[t]，及其總花費 TCost

首先，I[0]=i（PART1 讀檔的第 2 項），而 O[0]=D[0]=DC[0]=0；接著，T 期內各期應計算之成本(DC[t],t=1,2,…,T)包含存（缺）貨成本、訂貨成本以及製造成本。其中，

- 存（缺）貨成本：當期末存貨量(I[t])大於 0 時，則乘上單位存貨成本(如式(3)或(4))；反之，當存貨量為負值則表示有缺貨，因此將缺貨個數乘上單位缺貨成本(如式(5)或(6))；
- 訂貨成本：若當日有進行訂貨，則當日成本需加上單位訂貨成本(C)；
- 製造成本：單位製造成本乘上訂貨量(如式(3)或(5))；

最後，加總 T 期內所有日花費(DC[t],t=1,2,…,T)，記為 TCost(如式(7))。

$$DC[t]=\begin{cases} H*I[t]+C+M*Q[t], & if\ I[t]>0,\ Q[t]>0 \quad (3)\\ H*I[t], & if\ I[t]>0,\ Q[t]=0 \quad (4)\\ S*(-I[t])+C+M*Q[t], & if\ I[t]\le 0,\ Q[t]>0 \quad (5)\\ S*(-I[t]), & if\ I[t]\le 0,\ Q[t]=0 \quad (6) \end{cases}$$

$$TCost=\sum_{t=1}^{T} DC[t] \quad (7)$$

針對第一種用安全存貨量訂貨時機之機制（即 Plc[0]&Plc[2]），以函式

```
void GetCost1(Policy *Plc, int T, int K , int i, int *D, int S, int
              H, int C, int L, int A)  實作，計算 Plc.I[], Plc.O[],
              Plc.DC[]及 Plc.TCost
```

針對第二種用再訂購點訂貨時機之機制（即 Plc[1]&Plc[3]），以函式

```
void GetCost2(Policy *Plc, int T, int K , int i, int *D, int S, int
              H, int C, int L, int R)  實作，計算 Plc.I[], Plc.O[],
              Plc.DC[]及 Plc.TCost
```

再以函式 void Print_Policy(Policy Plc, int T, int *D)將 Plc[0],…,Plc[3]等四種機制中總成本最小者之 D[t]、O[t]、I[t]、DC[t]列印出來，t=0,…,T。

流程綜合說明

假設輸入檔名為 HW6_input.txt，其內容如第三、四頁所示，則程式會在讀完輸入檔之後（即 PART1 結束）開始產生 T 日的需求 D[1],…,D[T]（此即為 PART2）；之後，依 PART3.1 將 4 種機制 Plc[0],…,Plc[3]的相關 array 設定好，再依 PART3.2 規定之函式一一依不同機制的規則計算其第 t 日的 O[t]、I[t]、DC[t]及其 T 日總成本 TCost，最後印出如下：

```
Total Cost: Plc[0]= 51800, Plc[1]= 55750, Plc[2]= 26100, Plc[3]= 30550
Detailed info for best strategy is Plc[2]:
    t       D       O       I       DC
    ------------------------------------
    0       0       0       2        0
    1       2      12       0     4450
    2       3       0      -3     1200
    3       5       0       4      400
    4       4      12       0     4450
    5       6       0      -6     2400
    6       2       0       4      400
    7       5      12      -1     4850
    8       6       0      -7     2800
    9       2       0       3      300
    10      4      12      -1     4850
```

解題建議

- 老師估計初學者必須花至少 10 天才能將本題作好，因此請同學務必早日開始。
- 本題分成 3 個 PART 處理，這 3 個 PART 其實可以先分開做好。由於讀檔或函式傳 pointer/reference 等 PART1 的要求理應在上幾次作業有練習過，應該還好；PART2 必須先弄懂計算原理，而最沒有困難的應該是 PART3，因此老師建議同學可先將 D[]當成已知的某些值，以其為基礎先將 PART3 各機制的 O[]、I[]、D[]之關係推導出來（譬如先在紙上將本頁「流程綜合說明」的範例之 D[]當已知，自行將其餘數字推導出來），再將該推導之邏輯寫成程式；待 PART3 處理完，再處理 PART2 及 PART1，這樣的順序可能較快。

***作業繳交應注意事項**

1. 作業需要繳交電子檔以及書面(列印程式檔及程式結果)
2. 電子檔請於作業繳交截止時間以前上傳至 http://moodle.ncku.edu.tw
   - 2.1 請同學先建立一個資料夾，資料夾名稱為“**學號_hw6**”，例如學號為 R12345678，則資料夾名稱則為 **R12345678_hw6**
   - 2.2 將程式檔案名稱存為“**hw6.cpp**”，並將此程式檔案存於上述設立之**學號_hw6** 資料夾中
   - 2.3 最後將整個**學號_hw6** 資料夾壓縮成 zip 檔(**學號_hw4.zip**)，再上傳至 moodle 系統(！注意！：**請勿**將 cpp 檔 copy/paste 至 word 檔而上傳之)

書面作業請於 **2011/05/23 上課 5 分鐘內(09:15 前)**繳交至講台。

附錄:

以下附上四種機制使用第一頁資料所產生的結果,老師建議同學可先用這個結果推導驗證自己是否理解本題的要求。先假設10 天的需求量為D=[2,3,5,4,6,2,5,6,2,4]，如何可計算出O[t]、I[t]、DC[t]及 TCost

Plc[0] (安全存貨 & Q = Q_1 = 6)

t	D	O	I	DC
0	0	0	2	0
1	2	6	0	3250
2	3	0	-3	1200
3	5	0	-2	800
4	4	6	-6	5650
5	6	0	-12	4800
6	2	0	-8	3200
7	5	6	-13	8450
8	6	0	-19	7600
9	2	0	-15	6000
10	4	6	-19	10850

TCost = 51800

Plc[1] (再訂購點 & Q = Q_1 = 6)

t	D	O	I	DC
0	0	0	2	0
1	2	0	0	0
2	3	6	-3	4450
3	5	0	-8	3200
4	4	0	-6	2400
5	6	6	-12	8050
6	2	0	-14	5600
7	5	0	-13	5200
8	6	6	-19	10850
9	2	0	-21	8400
10	4	0	-19	7600

TCost = 55750

Plc[2] (安全存貨 & Q = Q_2 = 12)

t	D	O	I	DC
0	0	0	2	0
1	2	12	0	4450
2	3	0	-3	1200
3	5	0	4	400
4	4	12	0	4450
5	6	0	-6	2400
6	2	0	4	400
7	5	12	-1	4850
8	6	0	-7	2800
9	2	0	3	300
10	4	12	-1	4850

TCost = 26100

Plc[3] (再訂購點 & Q = Q_2 = 12)

t	D	O	I	DC
0	0	0	2	0
1	2	0	0	0
2	3	12	-3	5650
3	5	0	-8	3200
4	4	0	0	0
5	6	12	-6	6850
6	2	0	-8	3200
7	5	0	-1	400
8	6	12	-7	7250
9	2	0	-9	3600
10	4	0	-1	400

TCost = 30550