第一組

組員：會計四 B05702117 涂譽寶

會計四 B05702075 張文毓

會計四 B05702057 李季澄

目錄

一、演算法構想

首先我們在不考慮期初存貨與產能上限的情況下，針對每日完成品的需求量訂定各工作站的生產數量與每日原料購買量，這時各生產站的生產量皆僅是為了剛好滿足當天完成品需求而生產。

接下來從第一個生產日開始配置期初存貨，我們優先配置完成品存貨，若完成品存貨得以滿足當天的完成品需求，當天各站就都不開機生產。反之，則確認在考慮完成品存貨的情況下，當日還有多少的完成品需求需要被滿足。藉由各站間的轉換率推算出當日各工作站與完成品所需生產及採購的數量，再減掉期初存貨後，我們就能得到每日各站為了達成完成品生產需求的產量配置。

完成上述兩步驟後，我們會得到每日各個工作站為了達成完成品產量需求的產量分配，然而由於我們每日的完成品生產目標，僅是為了讓期初完成品存貨加上當天的生產量可以剛好達到給定的每日需求量，因此某些工作站可能會因為當日完成品需求過高而被分配到超出其產能上限的目標。為此，我們從最後一天的最後一個工作站開始判斷該站的目標生產量是否超過產能上限，若有超過則讓該站生產到其產能上限，則將超出的數量分配至上一日同一工作站生產。配置該數量的同時，當日與上一日各工作站的生產目標也會因為該差額重新配置。另外，因為我們是從最後一天的最後一個工作站開始檢查，所以假如某一站因為接收前一個工作站的產量需求而造成目標產量超出限制時，也能藉由此迴圈將超額的部分再往前一天配置。完成這部分後，我們會得到每日各站在接收其他工作站產能上限後所需要的生產目標，超出當天需求的部分就作為存貨補足下一天產能不足的部分。

最後針對原料採購的部分，從最後一天開始與前幾天比較採購成本加上存貨成本後，哪天購買原料是最為划算的，將該日所需要購買的所有原料都交由該最佳購買日期購買，以此類推找到所有原料的最佳購買日期。

二、主要函式介紹

allowcate_inventory()

在這裡函數裡，我們都會根據每天的需求盡可能的把初始存貨消耗完。舉例來說，第一天的需求為100，那我們就會去看這天有多少最終產品初始存貨，來判斷第一天實際需要產出的最終產品數量，假設這天的初始存貨是50，則我們還需要在第一天生產50的最終產品，再由這50的需求去往前推每一個工作站的產量，並且依序判斷該工作站有多少的初始存貨，如果初始存貨 > 該工作站的需求量，則從該工作站以前的每一站便不需要生產，並且把初始存貨量扣掉該工作站的需求量，成為下一天的初始存貨。

//////////////////////

Pseudocode:

if 當天需求大於當天最終產品初始存貨

```
{
    實際產量＝當天需求 - 初始存貨
    依序將上一個工作站的產量改成實際產量所需的原料或在製品
    if 工作站的產量 > 工作站初始存貨
    {
       工作站初始存貨 = 0
       工作站的產量 -= 工作站初始存貨
    }
    else
    {
       工作站產量 = 0
       工作站的初始存貨 -= 工作站產量
    }
}
else
{
    最終產品初始存貨 -= 當天需求
    當天不需生產
}
//////////////////////
```

allowcate_bottleneck()

分配好初始存貨之後，我們也重新更新了每一站的實際產量，而這時還需要判斷每一站是否超出產能限制，也就是這個函數的功能。

由於每一天所需的原料或產品都必須從當天或當天以前生產出來，所以在這個函數裡，我們首先從最後一天的每一個工作站逐一檢查其是否超過產能限制。如果超過了產能限制，我們便把超過的部分丟給前一天的同一個工作站，同時也把超過的那部分產量所需的原料或在製品也往前一天丟給相對應的工作站，也就是說到數第二天的同一個工作站，將會需要多生產一些留給最後一天用。而在最後一天檢查完之後，便往前一天重複上述的步驟。最後的結果將會使得每天的每一個工作站都是滿足產能上限的狀態，並且也滿足當天的需求量。

//////////////////////

Pseudocode:

```
for(i = 最後一天, i >0, i -- )
{
    for(j = 最後一個工作站, j >0, j --)
```

```
    {
        if 超過產能上限
            把超過的部分放到前一天（包含原料跟在製品）
    }
}
//////////////////////
```

bestBuyRaw()

最後，我們再依據原料的採購成品以及原料的存貨成本來決定該天所需要的原料應該要在哪一天買好，因為原料一定要在當天以前買好，所以我們也是從最後一天開始。大致的想法是，只要在當天之前的採購成本加上存貨成本比當天的採購成本低，我們就選擇他，並選擇最小的那天買原料。

```
//////////////////////
Pseudocode:
for(i  = 最後一天, i >=0, i--)
{
    for(j = i - 1, j >=0, j--)
    {
        if 第j天的（採購成本 + 原料存貨成本）最小
        就讓 第 j 天採購
    }
}
//////////////////////
```

三、測試資料範例

假設有一筆測試資料如下：

各站機台數：10　10　10

各機台每日產量限制：240　240　240

各站產能限制：2,400　2,400　2,400

各站機台開機成本：200　600　300

各站生產成本：20　10　10

各站轉換率：3　2　2

各站存貨成本：4　10　25

原料及各站初始存貨：1000　1000　2350　100

每日產量需求：1200　1500　800　2100　1000

每日原料採購成本：10　40　30　1　6

我們將透過前述的演算法，逐步規劃生產，以下為實際步驟：

（一）分配初始存貨

首先，我們從第三站的產量去設定各站的每日產量，結果如下：

	原料量 =>	第一站產量 =>	第二站產量 =>	第三站產量
Day 1	7,200	2,400	1,200	600
Day 2	9,000	3,000	1,500	750
Day 3	4,800	1,600	800	400
Day 4	12,600	4,200	2,100	1,050
Day 5	6,000	2,000	1,000	500

接著，我們將初始存貨從第一天開始往最後一天分配，並從第三站開始往原料分配。這麼做的目的是希望能盡量降低初始存貨的存貨成本。另外，前站的生產規劃受到後站需求影響，因此我們從最後一站開始分配存貨。

先分配完成品：

	原料量 =>	第一站產量 =>	第二站產量 =>	第三站產量
Day 1	6,000	2,000	1,000	500
Day 2	9,000	3,000	1,500	750
Day 3	4,800	1,600	800	400
Day 4	12,600	4,200	2,100	1,050
Day 5	6,000	2,000	1,000	500

再分配第二站的在製品：

	原料量 =>	第一站產量 =>	第二站產量 =>	第三站產量
Day 1	0	0	0	500
Day 2	1,400	700	350	750
Day 3	4,800	1,600	800	400
Day 4	12,600	4,200	2,100	1,050
Day 5	6,000	2,000	1,000	500

接著分配第一站的在製品：

	原料量 =>	第一站產量 =>	第二站產量 =>	第三站產量
Day 1	0	0	0	500
Day 2	0	0	350	750
Day 3	3,900	1,300	800	400

Day 4	12,600	4,200	2,100	1,050
Day 5	6,000	2,000	1,000	500

最後，分配原料：

	原料量 =>	第一站產量 =>	第二站產量 =>	第三站產量
Day 1	0	0	0	500
Day 2	0	0	350	750
Day 3	2,900	1,300	800	400
Day 4	12,600	4,200	2,100	1,050
Day 5	6,000	2,000	1,000	500

將原料都分配完畢後，我們開始檢查現行的生產規劃是否超出每日的產能限制。
由於只有前日能夠分擔後日的生產需求，我們從最後一天的最後一站往後開始檢查並將超出的產能逐步往前一天推，同時調整當天各站的產量。

二、檢查產能限制

分配完初始存貨後的產能：

	原料量 =>	第一站產量 =>	第二站產量 =>	第三站產量
Day 1	0	0	0	500
Day 2	0	0	350	750
Day 3	2,900	1,300	800	400
Day 4	12,600	4,200	2,100	1,050
Day 5	6,000	2,000	1,000	500

第五天的產量皆沒超過產能，因此直接看到第四天的第一站（超出產能限制 1,200）：

	原料量 =>	第一站產量 =>	第二站產量 =>	第三站產量
Day 1	0	0	0	500
Day 2	0	0	350	750
Day 3	8,300	3,100	1,700	850
Day 4	7,200	2,400	1,200	600
Day 5	6,000	2,000	1,000	500

第四天產能符合限制後，再看到第三天的第一站（超出產能限制 700）：

	原料量 =>	第一站產量 =>	第二站產量 =>	第三站產量
Day 1	0	0	0	500
Day 2	2,100	700	700	925

Day 3	10,400	2,400	1,350	675
Day 4	7,200	2,400	1,200	600
Day 5	6,000	2,000	1,000	500

三、考量原料採購成本

若前日的原料成本低廉，考慮存貨成本後，在前日先將後日的原料買好或許能降低成本。

	原料量 =>	第一站產量 =>	第二站產量 =>	第三站產量
Day 1	0	0	0	500
Day 2	2,100	700	700	925
Day 3	10,400	2,400	1,350	675
Day 4	7,200	2,400	1,200	600
Day 5	6,000	2,000	1,000	500

首先，我們以第五天為基準計算每天的採購加存貨成本。

第五天 6,000 * 6 = 36,000

第四天 6,000 * 1 + 6,000 * 4 = 30,000

第三天 6,000 * 30 + 6,000 * 4 * 2 = 228,000

第二天 6,000 * 40 + 6,000 * 4 * 3 = 312,000

第一天 6,000 * 50 + 6,000 * 4 * 4 = 396,000

由於第四天的採購成本加存貨成本較第五天的原料成本低，我們將在第四天購買第五天的原料：

	原料量 =>	第一站產量 =>	第二站產量 =>	第三站產量
Day 1	0	0	0	500
Day 2	2,100	700	700	925
Day 3	10,400	2,400	1,350	675
Day 4	13,200	2,400	1,200	600
Day 5	0	2,000	1,000	500

而後續的部分，由於從不同基準來看，各天成本差距成一定比例關係，前日成本皆比後日成本高，已無再比較之必要。

四、分工與心得

組員 A-演算法設計、程式撰寫

心得

我們每一位組員應該都是第一次設計演算法與處理成本最佳化的問題，因此在思考上造成了許多盲點。我們一昧地嘗試將各種成本與產量配置集中於一個演算法中考量，不僅在討論的過程中造成了許多問題，在執行上也造成了許多困難。期間有同學想以最短路徑的方式尋求最低的生產限制，但由於各節點的分枝過大，最後 Time Limit Exceed 而無法執行。在目前這個演算法生成後，我們也只將大部分的心思放在如何精進這個演算法，投入了過多的時間卻只發現在該架構下能改進的空間實在不多。

我認為這次的作業是一次很好的經驗教訓，以後在面對類似的問題時能使我更快速的篩選掉旁枝末微的細節，也學到在面對瓶頸時，該迅速的跳脫思維尋找解決方案，不被既有的框架給限制住，如此才能快速有效的解決問題

組員 B- debugging, 優化函數

心得

這次的報告是一個非常好的實戰經驗，也讓我們深深的體會到要馬上寫出一個最佳化的演算法是很困難的，甚至有時候不應該執著於只找出那個“唯一解”而是可以試試看搭配著不同的演算法，視情況不同用不同的方式解決。同時，我們也體會到，跳脫框架的思考有多麼重要，有時候自己想了一個方法之後，就很容易停留在同一個思維邏輯裡打轉，沒有辦法用另一個角度看待這個問題，也就錯失了很多更有創意或是有效的演算法。 整體而言，這次的經驗真的學到了很多，不只是一些抽象的思考方式得到進步，也學會避免許多在寫程式時容易遇到的小錯誤，或是一些實用的技巧等等。

組員 C-演算法調整、debugging

心得

這是我第一次開發這類型的專案。從一開始的分工其實就不是很好分配，在選擇演算法時也容易陷入不斷考慮更複雜的情況後，乃至討論到完全破局的情況。最後，我們選了兩個方式分頭進行，一邊試圖用最短路徑問題來找出極低的成本，另一邊則是用直觀的方式一步一步地考慮各種情況。

也是因為有這麼做，最後才能寫出一個可行的演算法。然而，現在回頭仔細看仍有發現不少能改善的地方，可惜因為一開始太執著要直接寫出一個極佳的演算法，最後沒有足夠的時間調整與優化現行的演算法。這次的作業給了我更多執行專案的經驗，雖然說很實用也很有趣，但實際上，我仍然不是很確定執行與分工才是最有效率的，這部分也有待我未來繼續花費時間及心力慢慢地摸索出來。