

## Branch-and-bound Programming Assignment #4

1. Write a branch-and-bound algorithm for the problem of scheduling with deadlines discussed in Section 4.3.2.
2. Develop a Java Program to implement your branch-and-bound algorithm.
3. Given an example with at least 10 jobs to show your program working properly.

請繳交 Java source code 和一份說明文件。The document must contain your algorithm and screenshots for running the example described in step 3。

無說明文件，以零分計算此作業。

## 介紹

---

branch and bound(分支定界)目標是找出滿足條件的一個解，所謂分支就是採用廣度優先的策略，以 Queue 佇列方式下去實作，在每個節點中拋棄不滿足約束條件的節點，故不繼續擴展該節點以下的子樹。以下會以 branch and bound 來實作 scheduling with deadlines 排程的問題。

## 演算法

---

首先將工作的 Profit 由大至小排序

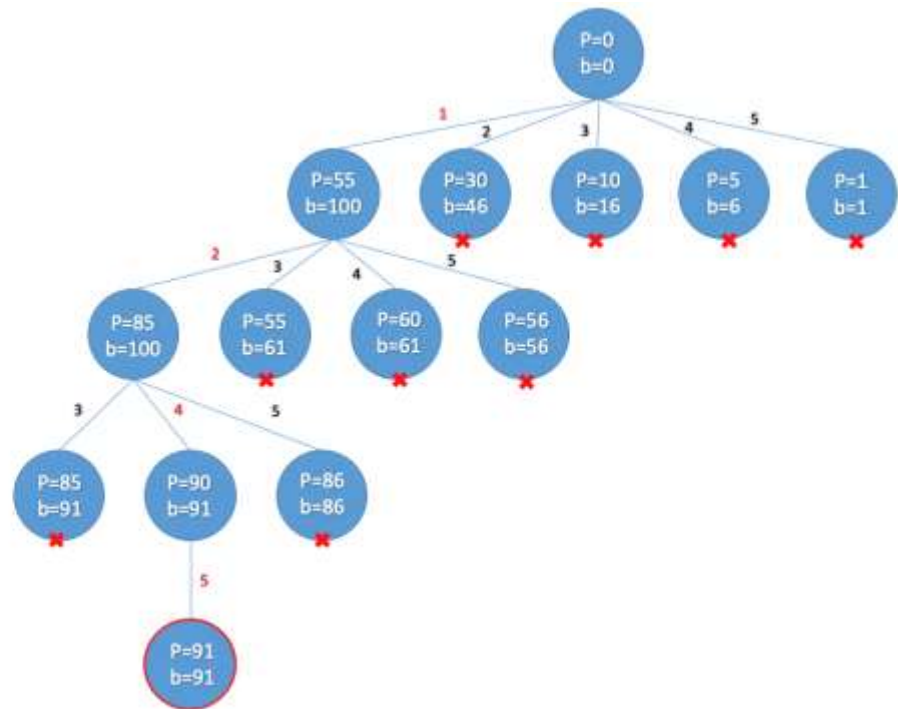
- Profit 算法:  
計算目前節點內可執行的工作 Profit 加總(需檢查 Deadline)
- Bound 算法:  
包含目前節點內可執行的工作尋找前 k 大的工作 Profit

以下範例是五個工作的空間狀態樹，Profit 已經先行排序(ps.壓縮檔內的 PPT 有完整計算過程)

Job	Profit	Deadline
1	55	1
2	30	2
3	10	1
4	5	3
5	1	4

Max Profit: 91

Job Sequence: 1 2 4 5



## Pseudocode

排程問題演算法架構如下，首先以 Profit 先行排序接著建立一個 Queue 佇列來儲存每個子節點 Node，並清空初始化佇列。接著依照廣度優先搜尋的順序逐一清空走訪佇列內的子節點。此外在迴圈中去比對每個工作的 deadline 是否已到若無代表該工作能進行並標記起來。計算 profit、bound 最後再比對 bound 是否小於等於 best，若成立則 nonpromising 該節點不繼續擴展。此外 best 為每次節點中的最大 profit，故每次計算完都要進行檢查並且更新 best。全部結束後若尚有子節點(promising)再將放入 Queue 中繼續走訪。

```

1  void scheduling() {
2      Sort(); // Sort by each Job's Profit
3      Queue < Node > PQ; // creat priority Queue
4      initialize(PQ);
5      while (!empty(PQ)) {
6          Node subNode;
7          Dequeue(PQ, subNode); // pull() and remove first node
8          for (each child nextSubNode of subNode) {
9              Node nextSubNode;
10             if (Job can be selected) {
11                 // checked the Job deadline legal or not
12                 // add up profit if deadline is legal
13             }
14             for(each job sequence){
15                 // checked each Job can be assign or not
16                 // calculate bound
17             }
18
19             if (bound is better than best) {
20                 // find the max profit
21                 best = profit;
22                 // promising and enqueue the child node
23                 enqueue(PQ, nextSubNode);
24             }
25         }
26     }
27 }

```

## 建立 scheduling 類別

為了資料方便處理這邊建立一個 scheduling 的類別專門放置每一個工作內容的利益和截止時間。該列別建立三個變數分別為字串型態的工作名稱(Job)，以及兩個整數型態的變數分別為利益(Profit)與截止時間(Deadline)。最後在建立建構子來初始化每個變數值。

```

1  class scheduling {
2      String job;
3      int profit, deadline;
4
5      public scheduling(String job, int profit, int deadline) {
6          this.job = job;
7          this.profit = profit;
8          this.deadline = deadline;
9      }
10 }

```

## 建立 Node 節點

我們必須將每個節點的資訊儲存下來，所以每個節點會有 bound、profit、list(目前工作串列)、arr(陣列儲存工作順序)、checked(陣列標記某個工作是否已經排定工作 0 與 1 表示)，最後自建立建構子初始化節點中的每一個變數。

```
1  class Node {
2      int bound;
3      int profit;
4      List<Integer> list;
5      int arr[];
6      int checked[];
7
8      public Node(int bound, int profit, List<Integer> list,
9      int arr[], int checked[]) {
10         this.bound = bound;
11         this.profit = profit;
12         this.list = list;
13         this.arr = arr;
14         this.checked = checked;
15     }
16 }
```

## main() 主程式

在主函式中主要目的是讀取使用者所輸入的測試資料首先要輸入整數 N 代表以下會有 N 個工作。接下來會要求使用者輸入 N 筆工作資料分別為(job name、profit、deadline)。

下圖程式第二行建立一個自訂義 scheduling 型態的 LinkedList 取名為 list 專門來儲存所有的工作項目與內容。第三行建立一個整數型態的陣列 solution[] 專門來儲存最佳解的工作順序。第四行有多個整數變數第一個 N 為工作數量。第二個 maxProfit 為所有工作序列中最大的利益值，並將它初始值為最小值，最後一個變數 maxDeadline 儲存工作序列中最大的截止時間(deadline)。

測資都依序輸入後我們先將這些資料以 Profit 進行排序，程式第二十三行使用第一個程式作業的合併排序來實作。

程式第二五行進入 Scheduling()函式使用 branch and bound 並使用最佳優先搜尋(Best-First Search)來尋找最佳工作排程以及計算最大利益。

最後印出結果，首先輸出排序後的工作序列，再來輸出滿足條件的最大利益，緊接著是該最大利益的一組解(工作序列)。

```

1 public class Main {
2     static LinkedList < scheduling > list;
3     static int[] solution;
4     static int N = 0, maxProfit = Integer.MIN_VALUE,
5     maxDeadline = Integer.MIN_VALUE;
6
7     public static void main(String[] args) {
8         Scanner scn = new Scanner(System.in);
9         list = new LinkedList < > ();
10        solution = new int[N];
11        System.out.print("請輸入工作數量: ");
12        N = Integer.parseInt(scn.nextLine()); // job count
13        System.out.println("請依序輸入Job Profit Deadline (以空白隔開):");
14        for (int i = 0; i < N; i++) {
15            // (job name, profit, deadline)
16            String arr[] = scn.nextLine().split(" ");
17            if (Integer.parseInt(arr[2]) > maxDeadline)
18                maxDeadline = Integer.parseInt(arr[2]);
19            list.add(new scheduling(arr[0], Integer.parseInt(arr[1])
20            , Integer.parseInt(arr[2])));
21        }
22        // (key要被排序資料,最左邊索引值,最右邊索引值,value值)
23        mergeSort(list, 0, list.size() - 1);
24        // 排程Branch-and-bound演算法
25        Scheduling();
26        System.out.println("\n| Job | Profit | Deadline |");
27        System.out.println("-----");
28        for (int i = 0; i < list.size(); i++) {
29            System.out.printf("%5s %8d %10d\n",
30            list.get(i).job, list.get(i).profit, list.get(i).deadline);
31        }
32        System.out.println("\nMax Profit: " + maxProfit);
33        // print selected job sequence
34        System.out.print("Job Sequence: ");
35        for (int i = 0; i < solution.length; i++) {
36            if (solution[i] != 0) {
37                System.out.print(list.get(solution[i] - 1).job + " ");
38            }
39        }
40    }
41 }

```

## Scheduling() 函式

---

此函式是使用最佳優先搜尋法來做排程運算，首先初始化 `best` 和 `bound` 變數。`profit` 變數是儲存每一次排程組合的利益值，`arr[]` 陣列是儲存每次工作排程的順序，`checked[]`陣列是紀錄目前某个工作是否已進入排程中 1 代表有排入工作，反之 0 尚未排入工作序列中。

此演算法是利用 Queue 佇列實作，採先進先出觀念(FIFO)，程式第八行採用 `while` 迴圈並判斷目前佇列中是否還有數值，直到佇列為空則跳出迴圈。程式第九行使用 `poll()` 方法用來取出佇列前端物件。第十三至十七行將所有變數初始化並取得上一個節點中計算出來的結果，二十二至二十九行計算 `profit` 加總並檢查是否可以執行此工作，若可以執行則將工作排成放入 `arr[]` 陣列中儲存並將此工作的利益加到變數 `profit` 中，最後在 `checked[]` 陣列中標註 1 代表此工作已被排定。程式碼三十至四十一行是計算 `bound`，`bound` 計算方式是包含目前節點內可執行的工作尋找前 k 大的工作 Profit 並加總。第四十二到四十八行首先判斷是否 `promising`，若是(`promising`)則更新 `best` 以及 `maxProfit` 接著繼續走訪子節點故將放入佇列中等待走訪(`branch`)，若 `profit` 小於 `best` 則確定了界限(`bound`)故不做後半部子樹走訪(`nonpromising`)。



```

1 public static void Scheduling() {
2     // initial variable
3     int upperBound = 0, bound = 0, profit = 0, arr[], checked[];
4     int count = 1;
5     Queue < Node > queue = new LinkedList < > ();
6     // offer()方法用來在佇列後端加入物件
7     queue.offer(new Node(0, 0, new LinkedList < > (), new int[N], new int[N]));
8     while (!queue.isEmpty()) {
9         Node subNode = queue.poll(); // poll()方法用來取出佇列前端物件
10        List < Integer > sublist = subNode.list; // 取得目前等待工作序列名單
11        // Best first search(最佳優先搜尋)
12        System.out.print(sublist + " ");
13        for (int i = 0; i < N; i++) {
14            bound = 0;
15            profit = subNode.profit;
16            arr = subNode.arr.clone();
17            checked = subNode.checked.clone();
18            if (sublist.size() == 0 || sublist.get(sublist.size() - 1) < i + 1) {
19                List < Integer > nextSubset = new
20                LinkedList < Integer > (sublist);
21                nextSubset.add(i + 1);
22                for (int k = list.get(i).deadline - 1; k >= 0; k--) {
23                    if (arr[k] == 0) {
24                        arr[k] = i + 1; // slot sign
25                        profit += list.get(i).profit;
26                        checked[i] = 1; // 目前工作被指派(標記1)
27                        break;
28                    }
29                }
30                int k = 0;
31                for (int j = 0; j < N; j++) {
32                    if (k == maxDeadline)
33                        break;
34                    if (j <= i && checked[j] == 1) {
35                        bound += list.get(j).profit;
36                        k++;
37                    } else if (checked[j] == 0 && j > i) {
38                        bound += list.get(j).profit;
39                        k++;
40                    }
41                }
42                if (profit > upperBound) {
43                    upperBound = profit;
44                    maxProfit = profit;
45                    solution = arr;
46                    // 寫入 queue (promising)
47                    queue.offer(new Node(bound, profit, nextSubset, arr, checked));
48                }
49            }
50        }
51    }
52 }

```

## 排序

在排程 Branch-and-bound 演算法過程中，資料要先行做排序才能執行。原理跟 01 背包類似，我們這邊是要尋找每個工作最大利益 Profit 故在我們的程式中首先要先以 Profit 由大至小來排序。以下程式碼為第一個程式作業合併排序，在此作業中套用。

```
1 public static void mergeSort(LinkedList < scheduling > list,
2 int left, int right) {
3     // 當左邊大於右邊時代表只剩一個元素了
4     if (left < right) {
5         // 每次對切，切到只剩一個為止
6         int mid = (left + right) / 2;
7         mergeSort(list, left, mid); // 左邊等份
8         mergeSort(list, mid + 1, right); // 右邊等份
9         Merge(list, left, mid + 1, right); // 排序且合併
10    }
11 }
12
13 public static void Merge(LinkedList < scheduling > list,
14 int left, int mid, int right) {
15     // 建立一個temp串列存放排序後的值
16     LinkedList < scheduling > temp = new LinkedList < > ();
17     int left_end = mid - 1; // 左邊最後一個位置
18     int index = left; // 位移起始
19     // 將最左邊的變數儲存起來(最後搬移元素會用到)
20     int origin_left = left;
21     for (int i = 0; i < right + 1; i++)
22         temp.add(new scheduling("", 0, 0));
23     // 左右兩串列比大小依序放入temp串列中儲存
24     while ((left <= left_end) && (mid <= right)) {
25         if (list.get(left).profit >= list.get(mid).profit)
26             temp.add(index++, list.get(left++));
27         else
28             temp.add(index++, list.get(mid++));
29     }
30
31     if (left <= left_end) { // 若左邊的串列尚未走完將剩餘的數值依序放入temp串列中
32         while (left <= left_end) {
33             temp.add(index++, list.get(left++));
34         }
35     } else { // 反之若右邊的串列尚未走完將剩餘的數值依序放入temp串列中
36         while (mid <= right) {
37             temp.add(index++, list.get(mid++));
38         }
39     }
40     // 最後將排序好的temp串列複製到list串列中
41     for (int i = origin_left; i <= right; i++) {
42         list.set(i, temp.get(i));
43     }
44 }
45 }
```



## 執行與測試

輸入說明：

第一行為工作數量  $N$ ，接下來會等待輸入  $N$  筆工作資料分別為 (Job Name、Profit、Deadline)。

輸出說明：

首先會輸出排序後的串列，接著輸出計算結果第一行為滿足條件的最大利益，第二行為該最大利益的一組解(工作序列)。

- 測試一

測資：

```
5
J1 55 1
J2 30 2
J3 10 1
J4 5 3
J5 1 4
55+30+5+1=91
```

```
<terminated> Main (7) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java (Dec
請輸入工作數量： 5
請依序輸入Job Profit Deadline (以空白隔開):
J1 55 1
J2 30 2
J3 10 1
J4 5 3
J5 1 4

| Job | Profit | Deadline |
-----|
| J1 | 55 | 1 |
| J2 | 30 | 2 |
| J3 | 10 | 1 |
| J4 | 5 | 3 |
| J5 | 1 | 4 |

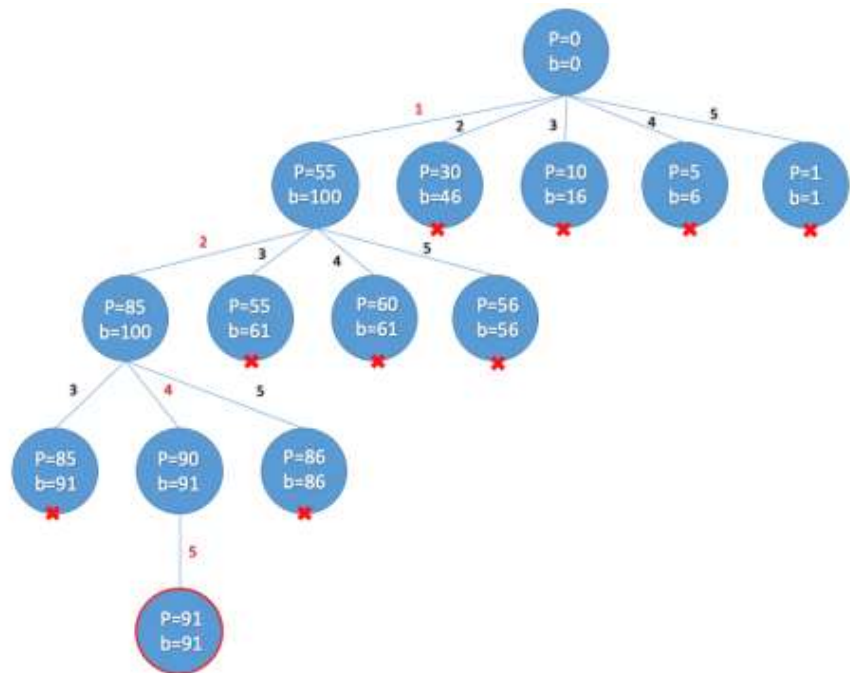
Max Profit: 91
Job Sequence: J1 J2 J4 J5
```

State Space Tree:

Job	Profit	Deadline
1	55	1
2	30	2
3	10	1
4	5	3
5	1	4

Max Profit: 91

Job Sequence: 1 2 4 5



## • 測試二

測資:

10

J1 80 1

J2 45 2

J3 40 3

J4 30 3

J5 30 2

J6 20 4

J7 10 2

J8 5 3

J9 4 1

J10 2 5

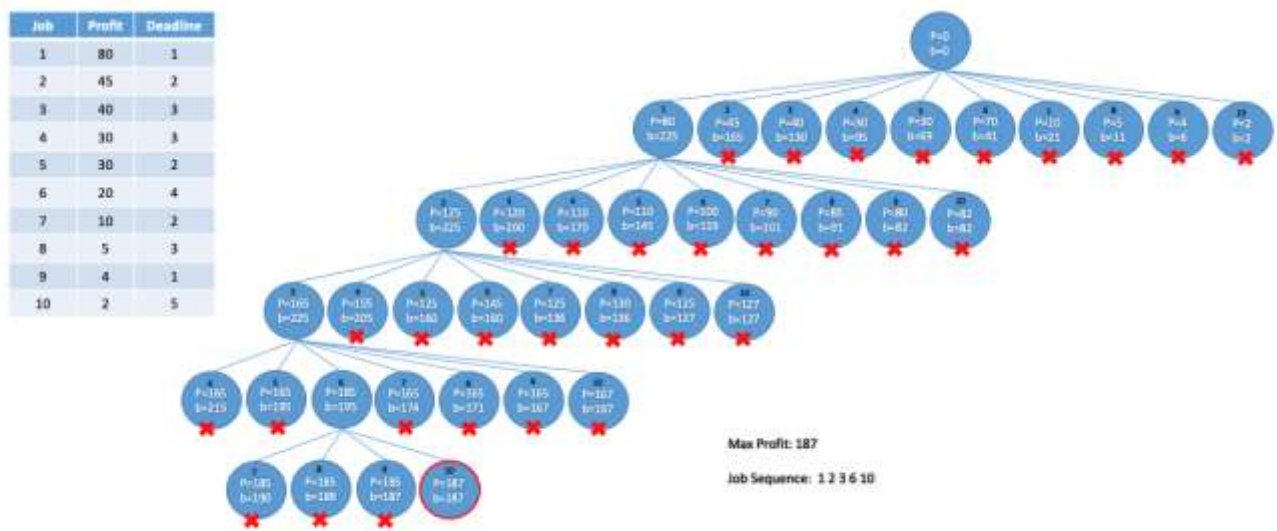
80+45+40+20+2=187

```

Problems Javadoc Declaration Console X
<terminated> Main (7) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java (De
請輸入工作數量: 10
請依序輸入Job Profit Deadline (以空白隔開):
J1 80 1
J2 45 2
J3 40 3
J4 30 3
J5 30 2
J6 20 4
J7 10 2
J8 5 3
J9 4 1
J10 2 5
|
| Job | Profit | Deadline |
|-----|
| J1 | 80 | 1 |
| J2 | 45 | 2 |
| J3 | 40 | 3 |
| J4 | 30 | 3 |
| J5 | 30 | 2 |
| J6 | 20 | 4 |
| J7 | 10 | 2 |
| J8 | 5 | 3 |
| J9 | 4 | 1 |
| J10 | 2 | 5 |
|
Max Profit: 187
Job Sequence: J1 J2 J3 J6 J10

```

State Space Tree:



- 測試三

測資：

```
10
J1 80 1
J2 2 5
J3 40 3
J4 10 2
J5 45 2
J6 20 4
J7 30 3
J8 5 3
J9 4 1
J10 30 2
```

$80+45+40+20+2=187$

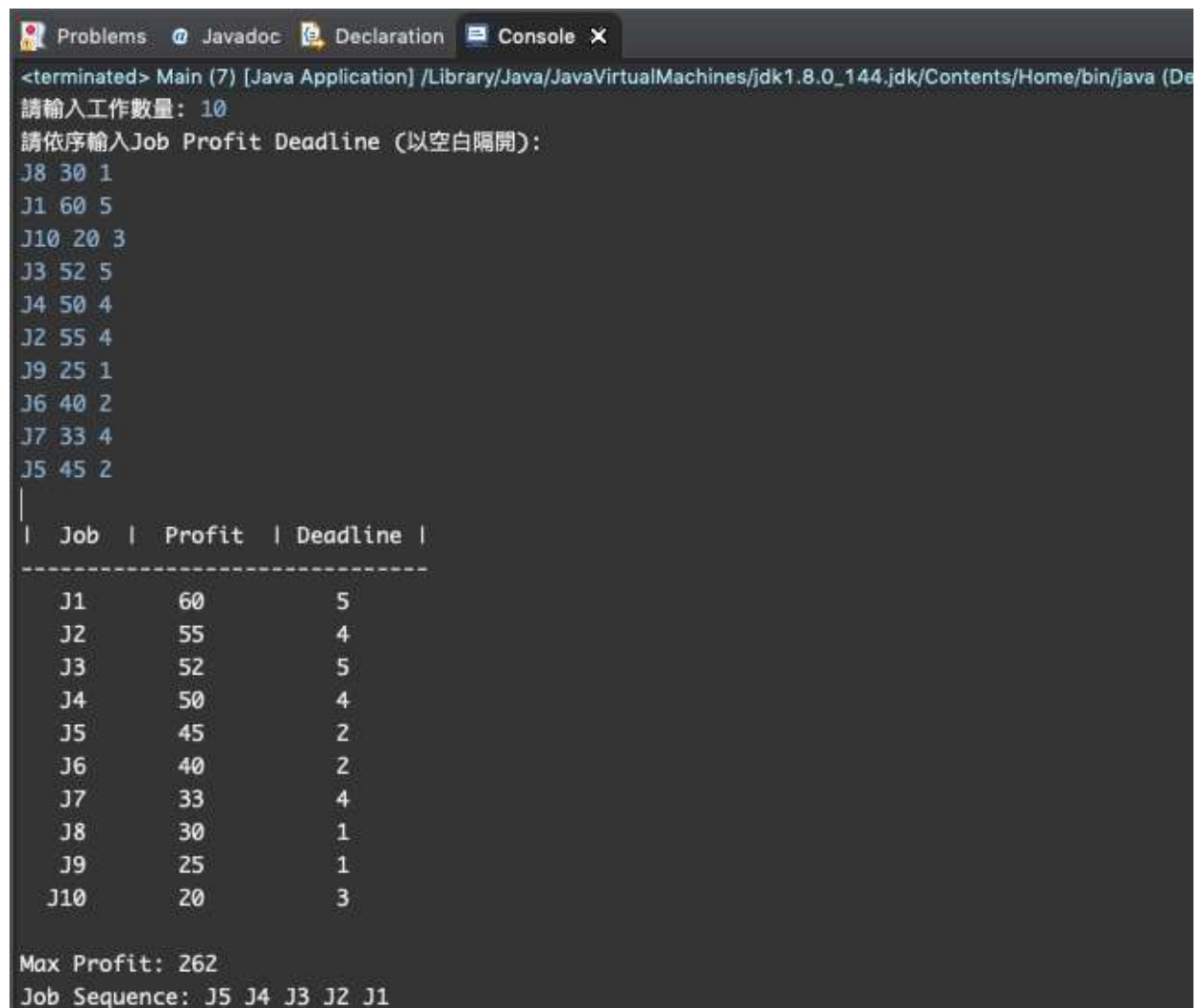
```
Problems Javadoc Declaration Console X
<terminated> Main (7) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java (De
請輸入工作數量： 10
請依序輸入Job Profit Deadline (以空白隔開):
J1 80 1
J2 2 5
J3 40 3
J4 10 2
J5 45 2
J6 20 4
J7 30 3
J8 5 3
J9 4 1
J10 30 2
|
| Job | Profit | Deadline |
|-----|
| J1   | 80     | 1       |
| J5   | 45     | 2       |
| J3   | 40     | 3       |
| J7   | 30     | 3       |
| J10  | 30     | 2       |
| J6   | 20     | 4       |
| J4   | 10     | 2       |
| J8   | 5      | 3       |
| J9   | 4      | 1       |
| J2   | 2      | 5       |
|
Max Profit: 187
Job Sequence: J1 J5 J3 J6 J2
```

- 測試四

測資：

```
10
J8 30 1
J1 60 5
J10 20 3
J3 52 5
J4 50 4
J2 55 4
J9 25 1
J6 40 2
J7 33 4
J5 45 2
```

$$45+50+52+55+60=262$$



```
<terminated> Main (7) [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_144.jdk/Contents/Home/bin/java (De
請輸入工作數量： 10
請依序輸入Job Profit Deadline (以空白隔開):
J8 30 1
J1 60 5
J10 20 3
J3 52 5
J4 50 4
J2 55 4
J9 25 1
J6 40 2
J7 33 4
J5 45 2
|
| Job | Profit | Deadline |
-----
J1      60      5
J2      55      4
J3      52      5
J4      50      4
J5      45      2
J6      40      2
J7      33      4
J8      30      1
J9      25      1
J10     20      3

Max Profit: 262
Job Sequence: J5 J4 J3 J2 J1
```