

Branch-and-bound Programming Assignment #4

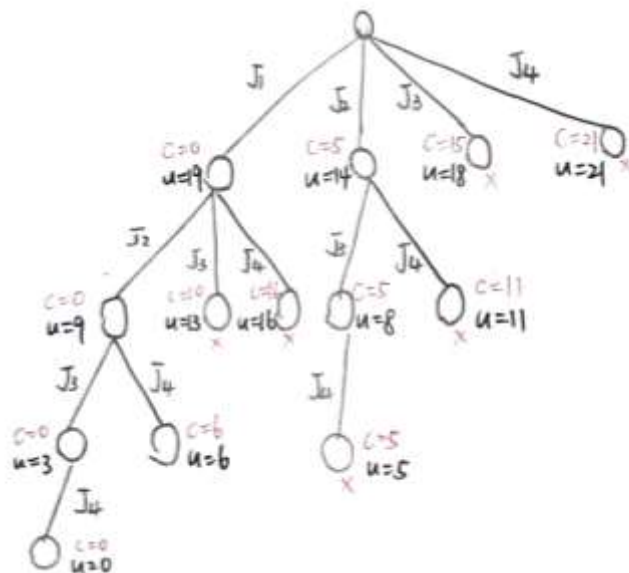
1. Write a branch-and-bound algorithm for the problem of scheduling with deadlines discussed in Section 4.3.2.
 2. Develop a Java Program to implement your branch-and-bound algorithm.
 3. Given an example with at least 10 jobs to show your program working properly.
- 請繳交 Java source code 和一份說明文件。The 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。

演算法

- cost 算法:
目前工作的子節點之前，尚未被指派工作的所有 profit 加總
- bound 算法:
除了自己本身工作之外，將所有尚未被排定工作的 profit 總和

Job	J ₁	J ₂	J ₃	J ₄
profit	5	10	6	3
deadline	1	3	2	1



排程問題演算法架構如下，首先建立一個 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 以及 cost 最後再比對 bound 是否小於 upperBound，最大上限為每次 bound 的最小值而且成本不能大於最大上限。全部結束後若有子節點再將它放入 Queue 中繼續走訪。

```
1 void scheduling() {
2     Queue < Node > PQ;
3     initialize(PQ);
4     while (!empty(PQ)) {
5         Node subNode;
6         dequeue(PQ, subNode);
7         for (each child nextSubNode of subNode) {
8             Node nextSubNode;
9             if (Job can be selected) {
10                // slot sign & mark the Job
11                // find the max profit
12            }
13            // calculate bound & cost
14            if (bound is better than upperBound)
15                upperBound = bound;
16            if (upperBound is better than cost)
17                enqueue(PQ, nextSubNode);
18        }
19    }
20 }
21 }
```

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。該類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(目前節點中的上限)、cost(目前節點中的所有成本)、profit(此節點中的利益)、list(記錄目前所有工作等待被排程)、arr(陣列儲存工作順序)、checked(陣列標記某个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 cost;
4      int profit;
5      List<Integer> list;
6      int arr[];
7      int checked[];
8
9      public Node(int bound, int cost, int profit, List<Integer> list, int
10     arr[], int checked[]) {
11         this.bound = bound;
12         this.cost = cost;
13         this.profit = profit;
14         this.list = list;
15         this.arr = arr;
16         this.checked = checked;
17     }
18 }
```

main()函式

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定義 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。

程式第十五行進入 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。

最後並印出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)。

```
1 public class Main {
2     static LinkedList < scheduling > list;
3     static int[] solution;
4     static int N = 0, maxProfit = Integer.MIN_VALUE;
5
6     public static void main(String[] args) {
7         Scanner scn = new Scanner(System.in);
8         list = new LinkedList < > ();
9         solution = new int[N];
10        N = Integer.parseInt(scn.nextLine()); // job count
11        for (int i = 0; i < N; i++) {
12            // (job name, profit, deadline)
13            String arr[] = scn.nextLine().split(" ");
14            list.add(new scheduling(arr[0],
15                Integer.parseInt(arr[1]), Integer.parseInt(arr[2])));
16        }
17        Scheduling();
18        System.out.println("max profit: " + maxProfit);
19        // print selected job sequence
20        for (int i = 0; i < solution.length; i++) {
21            if (solution[i] != 0) {
22                System.out.print(list.get(solution[i] - 1).job + " ");
23            }
24        }
25    }
26 }
```

Scheduling()函式

此函式是使用最佳優先搜尋法來做排程運算，首先建立 `cost` 變數來計算目前的成本，建立 `bound` 計算所有未被指派工作的 `profit` 加總，以及建立變數 `upperBound` 來決定目前最大的 `bound` 值，`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[]`陣列是紀錄目前某個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)，再搭配細部修改變成最佳搜尋演算法實例，程式第七行採用 `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。程式第六行使用 `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，二十一至二十八行計算 `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。程式碼三十至三十三行是比較取得目前最大的 `profit` 並記錄下來。程式碼三十四至四十三行是分別計算 `cost`(成本)，計算方式為目前被指派工作之前尚未被指派工作的 `profit` 加總；而 `upper` 計算方式為除了自己和已被排定的工作之外將所有未被指派工作的 `profit` 加總。第四十七行是判斷成本是否大於最大上限，若大於則確定了界限(`bound`)故不做後半部子樹走訪，反之繼續走訪子節點故將放入佇列中等待走訪(`branch`)。程式碼四十八至五十行是判斷目前最小的 `bound` 值，若 `bound < upperBound` 則放入 `upperBound` 中取代代表目前節點中的最大上限值作為後面判斷的依據。

```

1 public static void Scheduling() {
2     // initial variable
3     int cost = 0, upperBound = Integer.MAX_VALUE, bound = 0
4     , profit = 0, level = 0, arr[], checked[];
5     Queue < Node > queue = new LinkedList < > ();
6     queue.offer(new Node(0, 0, 0, new LinkedList < > (), new int[N], new int[N]));
7     // offer()方法用來在佇列後端加入物件
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        for (int i = 0; i < N; i++) {
13            cost = 0;
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 LinkedList < Integer > (sublist);
20                nextSubset.add(i + 1);
21                for (int k = list.get(i).deadline - 1; k >= 0; k--) {
22                    if (arr[k] == 0) {
23                        arr[k] = i + 1; // slot sign
24                        profit += list.get(i).profit;
25                        checked[i] = 1; // 目前工作被指派(標記1)
26                        break;
27                    }
28                }
29
30                if (profit > maxProfit) {
31                    maxProfit = profit;
32                    solution = arr;
33                }
34                for (int j = N - 1; j >= 0; j--) {
35                    // 計算 cost (成本)
36                    if (checked[j] == 0 && j < i) {
37                        cost += list.get(j).profit;
38                    }
39                    // 計算 bound (上限)
40                    if (j != i && checked[j] == 0) {
41                        bound += list.get(j).profit;
42                    }
43                }
44                // 成本不能大於最大上限
45                if (cost > upperBound)
46                    continue;
47                // 最大上限為每次 bound 最小值
48                if (bound < upperBound) {
49                    upperBound = bound;
50                }
51                // 寫入 queue
52                queue.offer(new Node(bound, cost, profit, nextSubset, arr, checked));
53            }
54        }
55    }
56 }

```

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：

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)。

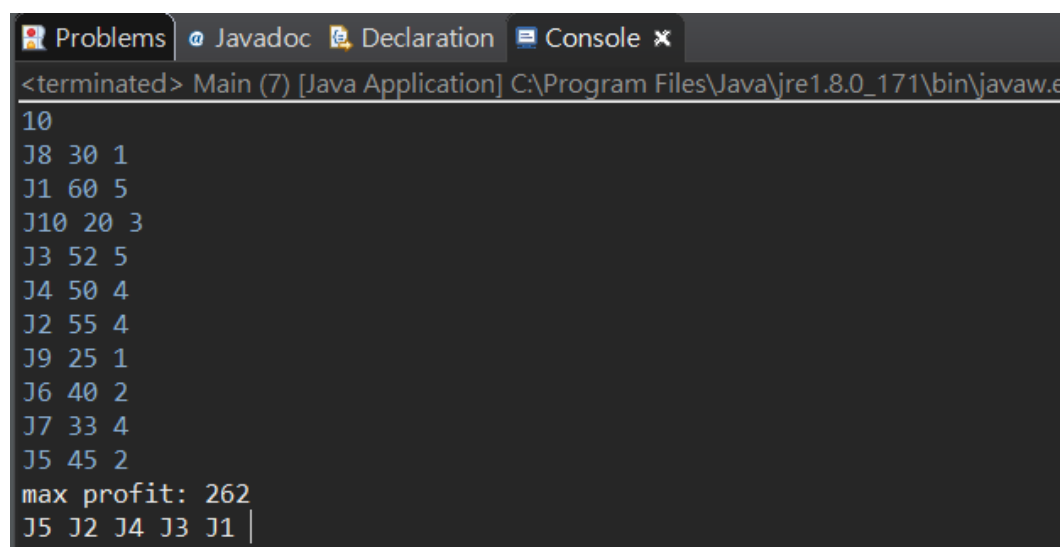
- 測試一

總共十個工作

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+55+50+52+60=262



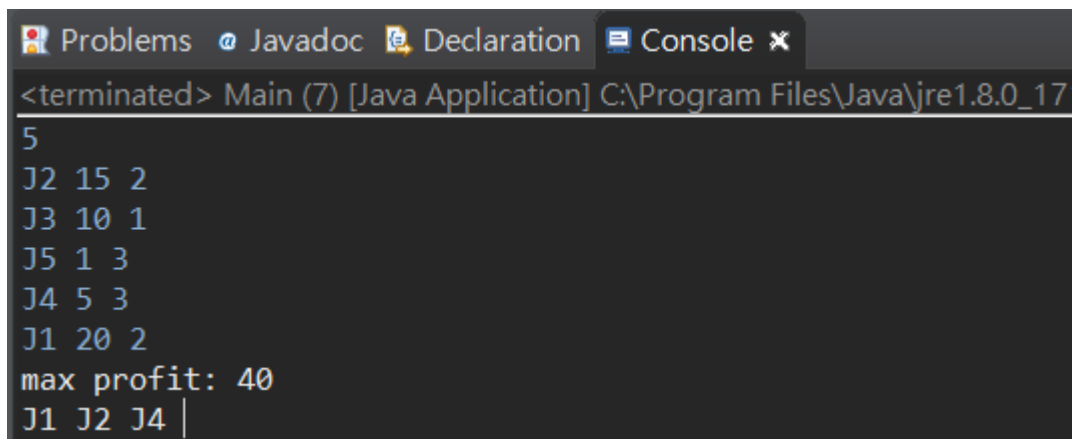
```
<terminated> Main (7) [Java Application] C:\Program Files\Java\jre1.8.0_171\bin\javaw.e
10
J8 30 1
J1 60 5
J10 20 3
J3 52 5
J4 50 4
J2 55 4
J9 25 1
J6 40 2
J7 33 4
J5 45 2
max profit: 262
J5 J2 J4 J3 J1 |
```

- 測試二

測資：

```
5
J2 15 2
J3 10 1
J5 1 3
J4 5 3
J1 20 2
```

$20+15+5=40$



The screenshot shows a Java IDE with a console window. The console output is as follows:

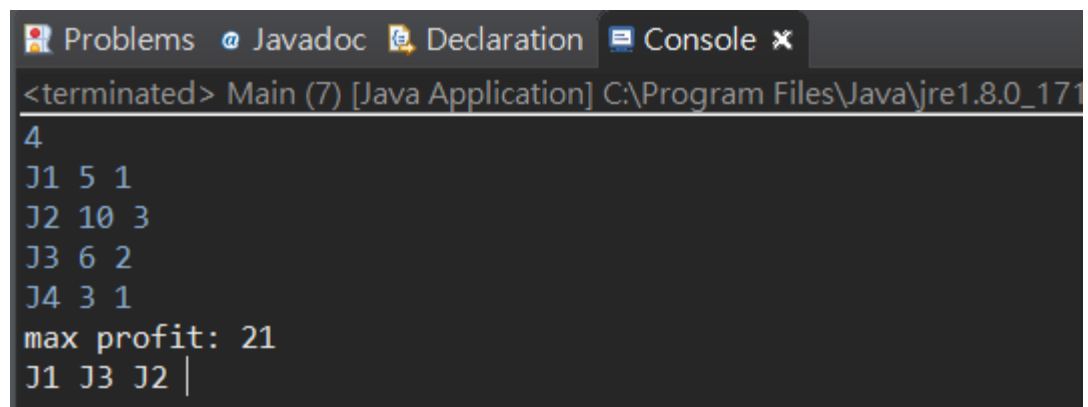
```
<terminated> Main (7) [Java Application] C:\Program Files\Java\jre1.8.0_17
5
J2 15 2
J3 10 1
J5 1 3
J4 5 3
J1 20 2
max profit: 40
J1 J2 J4 |
```


- 測試三

測資:

```
4
J1 5 1
J2 10 3
J3 6 2
J4 3 1
```

5+6+10=21



The screenshot shows an IDE console window with the following tabs: Problems, Javadoc, Declaration, and Console. The console output is as follows:

```
<terminated> Main (7) [Java Application] C:\Program Files\Java\jre1.8.0_171
4
J1 5 1
J2 10 3
J3 6 2
J4 3 1
max profit: 21
J1 J3 J2 |
```