COMP3010 Algorithm Theory and Design

Assignment 2

Last updated: September 13, 2020

1 Background

For your second assignment, we are going to look at a fairly well-known combinatorial optimisation problem which can be solved using dynamic programming. Therefore, the main aim for this assignment is to give you some experience in constructing a dynamic programming algorithm. Hopefully you can also see for yourself just how much faster it is compared to a brute-force approach.

To explain the problem, let us try to use a real-world example. Imagine that you are working for a real-estate developer and your company is in the process of selling a parcel of land to the public. The following picture is an example of the lots which are available (the image is taken from http://www.madorabay.com.au/, but please note that I am not affiliated with this company in any way, and I am not trying to promote the best value beachside land that they have for sale in case you plan to retire in Western Australia).

Figure 1: Madora Bay, available lots for sale, from http://www.madorabay.com.au/lots-for-sale

The public is allowed to purchase one or more lots, and anyone who wishes to make a purchase must submit a bid, detailing which lot(s) they are interested in and the amount of money they are willing to pay for it. Some lots are more valuable than others, so multiple parties may be interested in purchasing the same lot. In this case,

obviously the company would sell to the highest bidder. Of course, not all lots will have a bid on them, since some lots may be undesirable.

For your task, you will be given the listing of all the bids that have been submitted so far, and from this information, you are required to work out which bids the company should accept in order to maximise their income. The problem would be simple if everyone could only bid for one lot, but as described earlier, it is possible to purchase more than one lot, and in fact this happens quite often (people want to build bigger houses, businesses want to build bigger shops, lecturers want to make harder assignments).

2 Problem Description

For your assignment, we are going to simplify the problem a little bit. In Figure 1, the lots are placed in a nice configuration based on the contour of the land. However, we can just place all the lots on a straight line without really changing the problem, so that is what we are going to do. Plus, since we are computer scientists, we will number our lots from 0 instead of 1. In short, we can just represent the lots using a one-dimensional array.

Each party can submit multiple bids, but we will only be concerned with the actual bids, not who made the bid, so we can ignore this detail. Each bid is going to be assigned an identifier as they come in, starting from zero, so if there are n bids, they will be numbered from 0 to $(n-1)$. Each bid consists of three values: the number of the starting lot, the number of the ending lot, and the price the bidder is willing to pay (in bitcoins). You can assume that the bidder wants to purchase every lot from the starting lot to the ending lot. For example, if a bidder submits the bid

33 39 126

then the bidder is looking to purchase lots 33 to 39 (i.e. 7 lots) and is willing to pay 126 bitcoins for it.

If there are overlapping bids, then you must decide which bid to refuse and which bid to accept. You can assume all bids are final, as in, they will not change and there will not be any new bids coming in for unsold lots. For example, suppose there are 10 lots for sale and there are only three bids:

0 4 24
5 9 27
4 5 50

In the above example, the third bidder is willing to pay 50 bitcoins for 2 lots (25 bitcoins per lot), while the first and second bidders average around 5 bitcoins per lot. the company will still make more (51 vs 50 bitcoins) selling to the first two bidders, even though the price per lot is much lower.

3 Input and Output

The input for the problem will be given to you in a text file which you have to parse. Each file has two integers on the first line;

n b

where n is the number of lots available for sale and b is the number of bids that have been submitted. The next k lines contains four integers separated by space;

i start end price

respectively denoting the bid number, the starting and ending lot numbers, and the price the purchaser is willing to pay. For example, then input

```
15 6
```

```
0 0 5 13
1 10 14 14
2 0 2 8
3 7 12 18
4 4 7 10
5 8 12 8
```

means that there are 15 lots for sale with 6 bids (numbered from 0 to 5). The first bid (bid 0) is willing to pay $12 for the lots numbered 0 to 5, the second bid (bid 1) is willing to pay $14 for the lots numbered 10 to 15, and so on.

The output you need to produce is an integer value denoting the maximum income that the company can make based on this input. For the above example, the answer is 32, choosing bids 1, 2, and 4.

As another example, the input

```
15 7
0 0 1 9
1 1 1 14
2 2 2 14
3 0 4 20
4 1 9 20
5 2 12 20
6 6 6 7
```

should return 35 as the optimal answer, choosing bids 1, 2, and 6.

4 Code Template

A template and a set of test files have been made available on the COMP3010 GitHub page. (direct link is https://github.com/sutantyo/comp3010_2020/tree/master/codes/src/assg2). You should be able to do a git pull from Eclipse as before. The template is very similar to the one given to you for the first assignment. You need to work only on one file, Solver.java and implement two methods:

```
public void readData(String infile) throws Exception
```

which you should use to read the data from the input files and

```
public int solve(String infile)
```

which should call readData() and then find the solution to the input given in the file.

You may of course add any class attributes/variables and methods as you see fit, although as before, I still require you to put all your code in the file Solver.java. Please declare any classes that you use inside the Solver class.

5 Marking

The second assignment is worth another 10% of your total marks in COMP3010, and as before, it will be marked out of 10, coming from (subject to some minor changes):

- 4 marks from passing the JUnit test cases,
- 1 mark from code quality and commenting, and
- 5 marks from the documentation (see the next section).

Note that you need to pass some of the JUnit test cases in order to receive marks full marks for code quality. For commenting and code organisation, please follow the guidelines I have outlined in the first assignment.

6 Documentation

For the documentation, you need to explain how you used dynamic programming in order to solve this problem. In particular, you need to answer the following questions which covers the steps in developing a dynamic programming algorithm, as covered in the lectures (each question is worth one mark):

1. Show how you can decompose the problem into subproblems.
2. Show that there exists overlapping subproblems (you can use an example for this).
3. Show that the problem exhibits optimal substructure.
4. Show the recursive relation that gives the optimal solution, that is, the maximum income that the company can get. Explain how it works (examples would be good here as well).
5. Show how you can derive the solution that leads to the optimal solution, that is, how to identify the bids that lead to the maximum income. You do not need to write the code for it. You may use an example to demonstrate how your algorithm works.

Please keep the documentation to be about 3-6 pages (one page per question should be plenty). Please note that I do not require you to explain your code in the documentation, however, your code should reflect the recursive relation that you used to answer (4). Plus, I still expect you to properly comment your code so that I can understand it.

Note (added 13/9): To clarify what I said above, you do not need to explain your code in a sense that you do not need to explain which function does what in the documentation (although this should be made clear in the code itself, via comments). However, you technically still have to explain your algorithm by giving the decomposition of the problem into subproblems and the recursive relation to obtain the optimal solution. To make this more explicit, I added that you should explain the recursive relation.

You are free to do either the bottom-up or the top-down version, but this must be consistent with your code, e.g. you cannot explain the top-down version in your documentation but then write a bottom-up DP algorithm (the recursive relation would be different).

7 Submission

Please submit the file Solver.java and the PDF of your documentation in the provided submission box on iLearn (under Assessments tab). Please do not zip them, just submit two files.

If you need to write extra classes for your solution, then please make them an inner class, as I plan to only run Solver.java. Let me know if this is a problem.

8 Warnings and Restrictions

- You must not use any Java libraries outside of java.io and java.util in Solver.java
- You can reuse code from lectures, textbook, and even the web, but you must cite your reference and explain how they work in your report.
- Please do not plagiarise or work together in writing the code. You can discuss the general algorithm with your friends (I cannot do anything about that), but my rule is, as soon as you write up the code, you have to stop the discussion. There are more than a hundred ways to write the code for this assignment (how many variables

you use, how many methods you need, how you do the brute force), and there are only 60-odd students in this unit, so it can be very obvious if you are writing the code together.

9 Change Log

- 3 September: Draft assignment released
- 10 September: Fixed incorrect example
- 11 September: Added section on documentation and code template