

Coursework Autumn Semester 2019

Professor Steve Schneider
Department of Computer Science
University of Surrey

Note



- The latest version of this handout is available on SurreyLearn. Check that you are working with the latest available version before submission.
- Check your submission with the python script attached or from the course website.
- The coursework is individualised. – You need to run a python script (attached or from the course website) to get your individual sets of values.

Deadline.

Your submission has to be received in SurreyLearn by

Mon 18th November 2019 – 16:00.

Instructions

- Answer both questions.
- Marks are awarded on an all-or-nothing basis per sub-question/text field, marks are only given for fully correct answers per sub-question/text field.
- A python script `check_submission.py` is available on the course website or as an attachment here: . You will need to run that script on your submission to check it is well-formed ¹. If you get a warning/error from the script, rectify your submission.
- A Python script `individual_values.py` will generate your individual values for the two questions. It is available on the website or attached here: .

¹You will need to install the package `pdfminer.six`. You can do this with the command `python2.7 -m pip install --user pdfminer.six`

- The attached script can be run on the desktop machines in the Computer Science lab as follows:

```
python2 individual_values.py <URN>
```


where <URN> is your individual URN.

- In order to run this on your own computer, you will have to have Python 2 installed plus additional python packages such as **numpy** – Consult the web on how to install these.
- Marking will be automated, ie a python script extracts your answers, and checks them against template answers or tests some of their properties as appropriate.
- Feedback will consist in the following:
 - My marking script produces a report on which answers are not correct.
 - A detailed sample solution will be provided when marks are returned.
- Submit your complete PDF file on SurreyLearn. Filename: `coursework_<username>.pdf` where <username> is your username, eg xy0123.
- You need to use a current version of the *original* Acrobat reader to fill this form in. Other PDF viewers are likely not to allow you to save the document with any contents in the active text fields. No current Acrobat reader version is supported on Linux.
- **Do not flatten the PDF document** (eg by printing it to a file) as this will make it impossible to extract your answers automatically which will result in a zero mark. Check with the attached script whether your pdf is a valid submission.
- Answers that require floating point numbers need to have three digits after the decimal dot with rounding of the last digit.
- For the sake of fairness and so that all of you have exactly the same information, I can't answer individual question on the coursework by email – instead please submit them to the SurreyLearn forum, or ask me during the lecture.
- Lists of values will be entered as comma-separated list as follows:
 - Empty List:** Leave the corresponding field empty.
 - List with one item:** Just enter the item without any commas
 - List with more than one item:** separate by comma like `a,b,c,d`
- Exercises must be solved *individually*, without consultation with other students. Students are advised to read the exercise questions carefully.

Marks Marks can be gained as follows:

- Question 1: up to 50 marks
- Question 2: up to 45 marks
- Compliance with submission instructions: up to 5 marks.

Individualisation

This coursework is individualised based on your URN, that is you will all do the same coursework, but everyone gets different numbers to start from. In order to get your individual numbers for Q1 and Q2 you will need to run the script (attached ) under Python 2. It will give you a set of 8 2D-points for Q1 and a set of 8 intervals for Q2.

Failure to use your individual numbers as output by the script or manipulating the script will lead to loss of all marks.

Student Information

Enter the following information:

First Name:
Last Name:
URN:
Username:

Declaration of Originality

I confirm that the submitted work is my own work. No element has been previously submitted for assessment, or where it has, it has been correctly referenced. I have also clearly identified and fully acknowledged all material that is entitled to be attributed to others (whether published or unpublished) using the referencing system set out in the programme handbook. I agree that the University may submit my work to means of checking this, such as the plagiarism detection service Turnitin(R) UK. I confirm that I understand that assessed work that has been shown to have been plagiarised will be penalised.

Note that “plagiarism” also includes collusion between students.

I confirm that I abide by the Declaration of Originality:

1 2D Minimal Distance

This exercise is about applying one of the algorithms from the lecture on a small scale by hand to experience how a divide-and-conquer algorithm works. In your solutions, refer to the points given by their index i (as given by the script) rather than just by their coordinates. It is essential that you follow the algorithm precisely and exactly.

Consider the following algorithm:

Algorithm 1 Algorithm to calculate the minimal distance between 2D points

```
procedure MINDIST(list of points, sorted by  $x$ -coordinate.)  
  if one point in the list then  
    return infinity  
  else if two points in the list then  
    return Euclidean distance of the two points  
  end if  
  Split list of points into left and right halves by  $x$ -coordinate.  
  Calculate  $x$ -coordinate of separation line  $L$  between left and right halves as the  
  average of the  $x$ -coordinates of right-most point in the left half and the left most point  
  on the right half.  
   $\delta_1 = \text{MINDIST}(\text{left half})$   
   $\delta_2 = \text{MINDIST}(\text{right half})$   
   $\delta = \min(\delta_1, \delta_2)$   
  Take all points less than  $\delta$  distance from separation line  $L$ .  
  Sort these points by  $y$ -coordinate.  
  Scan these points in  $y$ -order and calculate Euclidean distance between each point  
  and next 11 points (if any) by index.  
  if any of these distances is less than  $\delta$  then  
    update  $\delta$   
  end if  
  return  $\delta$   
end procedure
```

Using this algorithm, find the *minimal distance* of points within a set given S of 2D-points. Obtain your personal set of points as follows (not using this set will lead to loss of all marks):

1. Download the Python2 script `individual_values.py` to one of the Linux machines in the Computer Science lab.
2. Run the script as described earlier.
3. The script will print a list of 8 points to the screen – this is the set S of points you need to calculate with. Multiple runs of the script will produce exactly the same number with a given URN.

4. **Note:** Assistance running the scripts will be given in the labs in Week 5 and 6.
5. Whenever we refer to the index i of a point, we always mean the **original** index of the point, that is the one in the list that this script is printing out.


Calculate the minimal distance within your individual set of points following the above algorithm and enter the following selected intermediate results into the form fields below:²

1. Comma-separated indices of points when sorted by x -coordinate: **Marks: 3**
2. x -coordinate of dividing line L in the first call to **MinDist** (first level of recursion):
 $L =$ **Marks: 4**
3. Comma-separated indices of points to the left of L : $P_L =$ **Marks: 1**
4. Comma-separated indices of points to the right of L : $P_R =$ **Marks: 1**
5. x -coordinates of the dividing lines L_L and L_R for the two calls to **MinDist** in the second level of recursion:
 $L_L =$ **Marks: 4**
 $L_R =$ **Marks: 4**
6. Comma-separated indices of points to the left of L_L : $P_{LL} =$ **Marks: 1**
7. Comma-separated indices of points to the right of L_L but left of L :
 $P_{LR} =$ **Marks: 1**
8. Comma-separated indices of points to the left of L_R but right of L :
 $P_{RL} =$ **Marks: 1**
9. Comma-separated indices of points to the right of L_R : $P_{RR} =$ **Marks: 1**
10. Comma-separated indices i of the points S_L that are in the δ -strip around L_L (leave empty if none): **Marks: 8**
11. Minimal distance δ_L between all points to the left of L : **Marks: 2**
 $\delta_L =$

²We will need to calculate more intermediate values (such as the δ s resulting for the different splits etc) – but I am asking only for the ones that are crucial for inferring whether you followed the right algorithm.

12. Comma-separated list of indices i of the points S_R that are in the δ -strip around L_R (leave empty if none): **Marks: 8**
13. Minimal distance δ_R between all points to the right of L : **Marks: 1**
 $\delta_R =$
14. Comma-separated list of indices i of the points that are in the δ -strip around L (leave empty if none): **Marks: 8**
15. Overall minimal distance $\delta =$ **Marks: 2**

2 Weighted Interval Scheduling

Apply the weighted interval scheduling algorithms from the lecture. Again use the provided script  to generate your individual set of intervals based on your URN.

Algorithm 2 Weighted Interval Scheduling Algorithm

Calculate the maximum weight set of compatible intervals:

```

# Core of the algorithm:
procedure CALCOPT( $n, p(1), \dots, p(n), v_1, \dots, v_n$ )
     $M[0] = 0$ 
    for  $i = 1 \dots n$  do
         $M[i] = \max(M[i - 1], v_i + M[p(i)])$ 
    end for
    return  $M$ 
end procedure

# Postprocessing
procedure POSTPROCESS( $n, M$ )
     $j = n$ 
     $S = \emptyset$ 
    while  $j > 0$  do
        if  $M[j] > M[j - 1]$  then
             $S = S \cup \{j\}$ 
             $j = p(j)$ 
        else
             $j = j - 1$ 
        end if
    end while
    return  $S$ 
end procedure

# run the algorithm
Sort the  $n$  intervals given by  $(s_i, f_i, v_i)$  by finish times  $f_i$  so that  $f_1 < f_2 < \dots < f_n$ 
Compute  $p(1), p(2), \dots, p(n)$ 
 $M = \text{CALCOPT}(n, p(1) \dots p(n), v_1 \dots v_n)$ 
 $S = \text{POSTPROCESS}(n, M)$ 
# S now contains the indices  $i$  of the selected intervals

```

As a reminder, $p(i)$ is the greatest index such that $f_{p(i)} \leq s_i$ or 0 if no such index exists. In other words, it is the index of the latest finishing interval whose finish time is still compatible with interval i 's start time. Obtain your personal set of intervals by running `individual_value.py`. Then by following the algorithm, calculate the set of intervals of maximum value.

Enter below intermediate results in the course of executing the algorithm:

$p(1) =$ Marks: 2

$p(2) =$ Marks: 2

$p(3) =$ Marks: 2

$p(4) =$ Marks: 2

$p(5) =$ Marks: 2

$p(6) =$ Marks: 2

$p(7) =$ Marks: 2

$p(8) =$ Marks: 2

$M[1] =$ Marks: 3

$M[2] =$ Marks: 3

$M[3] =$ Marks: 3

$M[4] =$ Marks: 3

$M[5] =$ Marks: 3

$M[6] =$ Marks: 3

$M[7] =$ Marks: 4

$M[8] =$ Marks: 4

Comma-separated list of names (single capital letter) as printed by script
`individual_values.py` of selected intervals: Marks: 3