School of Computing and Information Systems

comp20005 Engineering Computation
Semester 1, 2018
Assignment 2

Learning Outcomes

In this project you will demonstrate your understanding of structures and arrays of structures, and will develop a computational solution for a non-trivial problem. You are also expected to make extensive use of functions; and to demonstrate that you have adopted a clear and elegant programming style. You will find it difficult to create a working solution unless you plan your program carefully in advance, and develop it incrementally.

Spatial Data

Much of the data that is used in computing (including engineering) applications is *spatiotemporal*, and describes the layout of objects in space and/or time. In this project we are going to pretend to have expert knowledge in the area of forest and tree management, and simulate the possible consequences of differing amounts of rainfall on the ability of a grove of trees to survive. To keep things manageable, and to allow straightforward output formats to be employed, many of the aspects of the required simulation are greatly simplified. Nevertheless the underlying programming principles – use of two dimensional arrays and graphical output visualizations – are important ones to master.

Input file `test1.tsv` describes five trees in a tab-separated input format with one header line:

```
label   xloc    yloc    liters  rootrad
A       14.8    23.8    18500   8.0
G       18.8    28.1    20800   7.0
F       24.1    22.2    31000   6.2
C       35.3    19.9    28000   7.3
E       16.5    10.5    15000   4.2
```

For example, the third tree has the label `F`, has its trunk at location 24.1 meters east of the origin (positive x direction) and 22.2 meters north of the origin (positive y direction), requires $31{,}00$ liters of water per year to survive, and sends its roots out in all directions to a radius of 6.2 meters from the location of its trunk. Each input file will contain at most 52 trees, with labels restricted to the letters `'a'` to `'z'` and `'A'` to `'Z'`. The trees are in no particular order in the file.

Stage 1 – Reading the data (marks up to 6/20)

In this stage you should read all of the data into internal structures suitable for use in the later stages. The required output of this stage is:

```
mac: myass2 < test1.tsv
S1: total data lines   =     5 trees
S1: total water needed = 0.113 megaliters per year
```

Hint: you should be able to get this stage going quite quickly starting from either your solution to the first project, or the sample solution to the first project. In either case you should acknowledge the source of any code that you have reused, even if it is your own.

Stage 2 – Multiway processing (marks up to 12/20)

Define the *catchment zone* of a tree to be a circle centered on its trunk, of radius given by its `rootrad` value. This is the area from which the tree is able to draw water. Two trees are in *conflict* if their catchment zones overlap. In this case their roots may be interlinked and they are competing for rain water.

In this stage you are to check each tree in turn, and for each of them list all of the other trees in the input with which they are in conflict. The required additional output for `test1.tsv` is:

```
S2: tree A is in conflict with G F
S2: tree G is in conflict with A F
S2: tree F is in conflict with A G C
S2: tree C is in conflict with F
S2: tree E is in conflict with
```

Distances between points should be calculated using the standard Euclidean formula: $d(x_1, y_1, x_2, y_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. You will need to include `math.h`, and use the `-lm` flag when compiling on `dimefox`: `gcc -Wall -lm -o myass2 myass2.c`

Stage 3 – Make a map (marks up to 16/20)

Now let's have some real fun. In order to help the arborists look after the trees, they would like a sketch map of where the trees are located and where the conflict regions are. Suppose that the region currently of interest to the arborists is 70 meters wide (in the east-west direction) and 60 meters tall (north-south direction), with the $(0, 0)$ origin at the south-west corner. To generate the plot, form a grid that is 70 cells/characters wide, and 30 characters tall (because printed characters are approximately twice as tall as they are wide). For example, the bottom-left cell in the grid is to represent the 1 meter wide (that is, in the east-west direction) by 2 meter tall cell (that is, in the north-south direction) of land from $[0 \leq x < 1] \times [0 \leq y < 2]$, and has its center at the point $(0.5, 1.0)$.

By considering the center point of every cell in the grid in turn, and asking (a) which of the trees that center point is closest to, and (b) whether that center point is within the catchment zone of that tree, a character plot can be built up showing which tree "claims" that cell as part of its catchment. Note that where there is conflict at the center point of a cell, the nearest tree claims the *whole* of that cell. When all cells have been considered and assigned (or not) to a tree, a plot can be output. For `test1.tsv` the top third of the plot is all blanks, and (only) the bottom two thirds of the required output is as shown in Figure 1. Full examples are linked from the Assignment FAQ page.

Note carefully the layout of the axes. The numbers that are printed indicate the x and y coordinates of the lower-left (south-west) corner of the cells. There will thus never be any cell labels plotted in the top row indicated by the `60 +`, and in the right-hand column, above the `70`. The character plotted in each cell is the label of the tree that is closest to the midpoint of that cell, or blank if the midpoint of the cell is not within the catchment zone of any of the trees. In all cases, it is assumed that if a cell is matched with a tree label, then all of any rainfall into that cell (totaling 2 square meters) is drawn to the corresponding tree by its roots.

You should also assume that all of the trees that are specified in any input file will lie fully (including their complete catchment zone) within the designated plot zone of $[0 \leq x < 70] \times [0 \leq y < 60]$. But note that the numbers 60 and 70 should be `#define`'d, and your program should operate correctly if they were altered to other values such as 50 and 60, provided that the input trees all fall in that smaller rectangle.

Stage 4 – Let it rain! (marks up to 20/20)

If rainfall of one millimeter occurs on one square meter of land, a total of one liter of water is available. In this stage, your program should access a total annual rainfall value in millimeters from the commandline

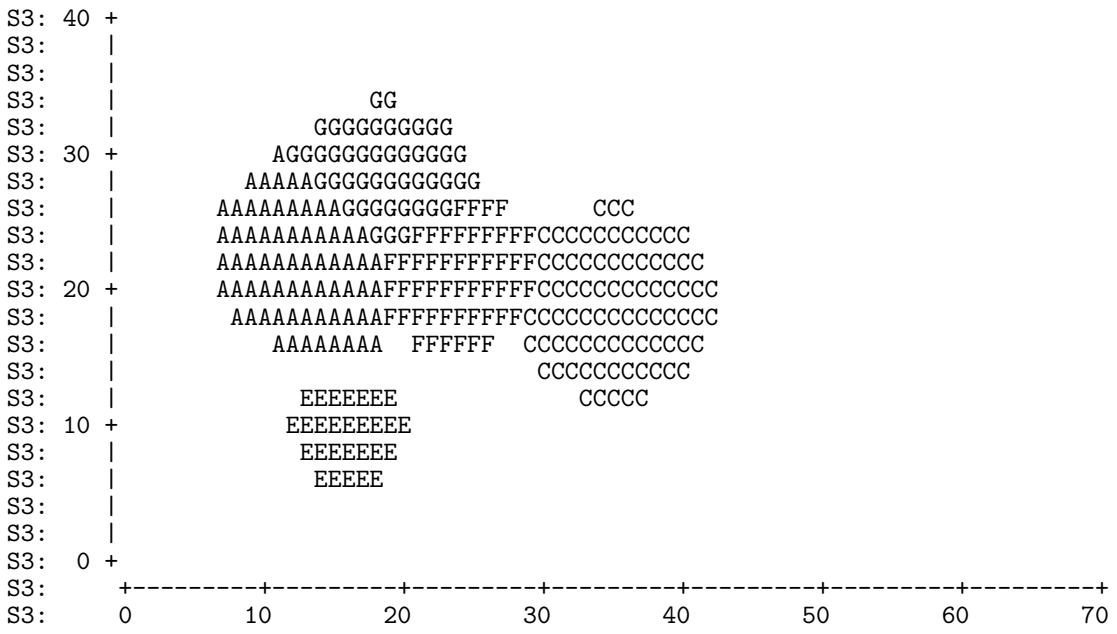

Figure 1: Partial output for Stage 3 for `test1.tsv`. See the LMS for full examples.

used to run the program, and then check to see (using the reduced catchment zones calculated in Stage 3, including handling the conflicts via the cell-at-a-time approximation process) whether each tree will be able to survive based on that much rainfall. For example, referring again to `test1.tsv` and Figure 1, tree `E` has a total of 28 cells associated with it, or 56 square meters. Because tree `E` has no conflicts, its computed catchment zone is a reasonable approximation of the circular area $\pi r^2 = 55.42$ square meters. For the other trees in this example the catchment zones are reduced because of conflicts – for example, tree `G` has a catchment of 48 cells, or 96 square meters, much less than the circle defined by its root radius (which would be 153.9 square meters). Returning to tree `E`: it needs 15,000 liters per year, which means (following the calculations you are to perform) it can survive provided the annual rainfall is greater than $15{,}000/56 = 267.9$ millimeters. (As indicative reference points, Melbourne's annual rainfall is about 650 millimeters, and Sydney's is 1200.)

Define the *stress factor* of a tree to be the ratio between the required-for-survival rainfall and the rainfall amount provided on the commandline. Trees with a stress factor greater than 1.0 will die; and in our simulation, they are to die in decreasing stress order. That means the death of one tree might reduce the stress on nearby trees once the conflicts are removed, allowing those other trees to shift to a lower stress factor, and to live.

To complete this stage, compute the stress factor for each tree for the rainfall amount given on the commandline, and if all trees have a stress factor of less than or equal to 1.0, then no trees will die and the process can finish. But if any trees have stress factors greater than 1.0, your program should identify the tree with the highest stress factor, and note it as having "died". You must then recalculate all of the catchment zones, assuming that dead trees draw no water, and once that is done, all of the stress factors. The process should then be repeated, at each iteration "killing" the tree with the highest stress factor greater than 1.0, and stopping once either all of the trees are dead, or when every surviving tree has a catchment that allows it to survive on the amount of rainfall that was specified on the commandline.

The output of this stage should report the sequence of trees that have died, and then draw another plot to illustrate the final tree arrangement. See the examples linked from the FAQ page for full details of what is required.

Modifications to the Specification

There are bound to be areas where this specification needs clarification or correction. Refer to the FAQ page at `http://people.eng.unimelb.edu.au/ammoffat/teaching/20005/ass2/` regularly for updates to these instructions. There is already a range of information provided there that you need to be aware of, with more to follow.

The Boring Stuff...

This project is worth 20% of your final mark. A rubric explaining the marking expectations will be provided on the FAQ page.

You need to submit your program for assessment; detailed instructions on how to do that will be posted on the LMS once submissions are opened. Submission will *not* be done via the LMS; instead you will need to log in to a Unix server and submit your files to a software system known as `submit`. You can (and should) use `submit` **both early and often** – to get used to the way it works, and also to check that your program compiles correctly on our test system, which has some different characteristics to the lab machines. *Failure to follow this simple advice is highly likely to result in tears.* Only the last submission that you make before the deadline will be marked.

You may discuss your work during your workshop, and with others in the class, but what gets typed into your program must be individual work, not copied from anyone else. So, do **not** give hard copy or soft copy of your work to anyone else; do **not** "lend" your "Uni backup" memory stick to others for any reason at all; and do **not** ask others to give you their programs "just so that I can take a look and get some ideas, I won't copy, honest". The best way to help your friends in this regard is to say a very firm "**no**" when they ask for a copy of, or to see, your program, pointing out that your "**no**", and their acceptance of that decision, is the only thing that will preserve your friendship. *A sophisticated program that undertakes deep structural analysis of C code identifying regions of similarity will be run over all submissions in "compare every pair" mode. Students whose programs are so identified will be referred to the Student Center for possible disciplinary action without further warning. This message **is** the warning.* See `https://academicintegrity.unimelb.edu.au` for more information. Note also that solicitation of solutions via posts to online forums, whether or not there is payment involved, is also taken very seriously. In the past students have had their enrolment terminated for such behavior.

Deadline: Programs not submitted by **10:00am on Monday 21 May** will lose penalty marks at the rate of two marks per day or part day late. Students seeking extensions for medical or other "outside my control" reasons should email `ammoffat@unimelb.edu.au` as soon as possible after those circumstances arise. If you attend a GP or other health care professional as a result of illness, be sure to take a Health Professional Report form with you (get it from the Special Consideration section of the Student Portal), you will need this form to be filled out if your illness develops in to something that later requires a Special Consideration application to be lodged. You should scan the HPR form and send it in connection with any non-Special Consideration assignment extension requests.

Marks and a sample solution should be available on the LMS by Monday 4 June.

Disclaimer: None of the "facts" about trees and their water requirements given here are reliable, and it is all just a story I made up to motivate the project. In particular, when it rains, the water runs downhill, making land contours an important additional factor, and different soils and undergrowth have different absorption rates. As well, trees need varying amounts of water according to the temperature, and their size, and their age, and the growing season, and etc, dozens of different factors; and trees can also sometimes withstand one or more drought years before becoming eventually threatened. They also sometimes get too much water and drown to death. Indeed, this project description is misleading in so many different ways that a horticulture student would laugh themselves silly if they read it. Hopefully you don't know any horticulture students, and will just have fun writing the program.