YSC2229: Introductory Data Structures and Algorithms

Final Project

April 2019

Your final project will consist of two parts:

- a set of algorithmic tasks requiring implementation,
- an individual report.

This manuscript provides the detailed descriptions of the tasks, the rules for their completion in a team, as well as break-down of the grades. Please, submit your solutions (the archive with the project code and the report) via Yale-NUS Canvas.

1 Problem Set

Maximal grade for this part: 20 points (+ up to 2 peer-awarded points)

The Rules of the Game

To account for different group sizes, this final project comes with a ***new set of rules*** for grading the programming submissions.

This part of the assignment features three problems, each coming with an estimated workload of one or two persons. Within each group, the problems should be split evenly between the team members, according to their workload. For example, in a group of three people, two students might team up to solve Problem 1.3, and the remaining one will take care of, *e.g.*, Problem 1.2 or Problem 1.1. A team of four students should have the capacity to cover all of the offered problems.

In this project, each of the team members should *assign* themselves a task to handle and indicate this explicitly in the individual report. Some tasks can be handled jointly by a sub-team of two students.

It is *not* required that for a team to submit all the solutions to account for the *full workload capacity* (*e.g.*, all four problems for a team of four people). For instance, a team of three students might end up submitting only two-person-worthy workload. The grades will be thus awarded based on the *assignments* of problems to the team members, indicated in the individual reports.

For example, consider a team of four members: **A**, **B**, **C**, and **D**. Assume that **A** takes care of Problem 1.1, **B** handles Problem 1.2, and **C** and **D** work together on Problem 1.3. At the end, **A**, **C** and **D** put their solutions to the joint codebase to be submitted as a solution, and also indicate in their reports which problems they tackled, while **B** does not make a submission. In this case, **A**, **C**, and **D** will be graded on their parts of the submission (and, ideally, *each* awarded 20 points), while **B** will not be awarded any points. In a case if all team members contribute to the code base, the penalty for bugs will be applied in accordance with the task assignments. For instance, **A**, **C**, and **D** will not be penalised for the bugs in **B**'s solution.

Peer-awarded points

For this part of the project, the in-team interactions and discussions of problems are encouraged. Each student has two bonus points (not counted towards their grade) that should feel free to award their team members (this should be indicated in the individual report). For instance, if **B** feels that they benefited from discussions with **A**, and **D** helped them with testing, **B** might decide to award, *e.g.*, one point to **A** and one point to **D**. It is not required to award any point to the team peers, and also it is allowed to award both points to the same person (as long as it's not yourself). Non-awarded points will vanish.

1.1 A Century Problem

Workload: One person

Consider a problem of listing *all* the ways the operations + and × can be inserted into the list of nine digits [1 . . . 9] so as to make a total *target* of 100. Two such ways are:

$$100 = 12 + 34 + 5 \times 6 + 7 + 8 + 9$$
$$100 = 1 + 2 \times 3 + 4 + 5 + 67 + 8 + 9$$

Note that **no parentheses** are allowed in expressions and × has a higher priority than +.

1. Devise an implement a data type to represent *candidates* for solving the century problem. Such candidates don't have to necessarily deliver 100. Examples include 12×345+67×89 and 1×2×3×4×5×6+789.
2. Implement a pretty-printer from your candidate data type to arithmetic expressions (as above).
3. Using your data type for candidates, construct a procedure that enumerates all of them. Use it to solve the century problem and deliver all expressions that solve the century problem. Test that those are indeed the solutions.

   Hint: You might want to experiment with smaller lists of digits (*e.g.*, $[1; 2; 3; 4]$) and smaller targets (*e.g.*, 10) to test your enumerations and the solver.

1.2 Sentinels in Graphs

Workload: One person

Definition 1 (Rooted directed graph) *A rooted* directed graph *is a triple $G = \langle V, E, v_0 \rangle$, where V is a set of nodes, $E \subseteq V \times V$ is a set of directed edges, and $v_0 \in V$ is a designated initial node, such that each other node is reachable from v_0 (by walking along the edges).*

Definition 2 (Paths) *Given a fixed graph G, we use the notation $u \longrightarrow v$ to indicate the edge $\langle u, v \rangle \in E$. A non-empty* path *$\sigma$ in a graph G is a sequence of nodes $\sigma = u_0, \ldots, u_n$, such that for all $i \in 1, \ldots, n$, $u_{i-1} \longrightarrow u_i$. Given a graph $G = \langle V, E, v_0 \rangle$, all finite paths starting from v_0 can be obtained by "walking through the set of its edges".*

Definition 3 (Sentinels in a rooted directed graph) *Given a graph $G = \langle V, E, v_0 \rangle$, a node u is a sentinel for a node v, if u belongs to* every *path from the initial node v_0 of the graph to v. The sentinel of v_0 is taken to be v_0.*

Definition 4 (Strict sentinel) *A node u is a strict sentinel for a node v if u is a sentinel for v and $u \neq v$.*

Definition 5 (Immediate sentinel) *The immediate sentinel for a node v is the unique node u that is a strict sentinel for v but is not a strict sentinel for any other node that is a strict sentinel for v. Every node, except the entry node v_0, has an immediate sentinel.*

Definition 6 (Sentinel tree) *A sentinel tree for a graph G is a tree where each node's parent is its immediate sentinel in G. Because the immediate sentinel is unique, it is a tree, and the start node v_0 of the graph is its root.*

In this part of your project we will be considering the *Sentinel Problem*, requiring one to find, for each node in the graph G, the set of nodes that are its sentinels, and the overal sentinel tree of G. For example, consider the graph in Figure 1. Nodes a, b, and c are sentinels for the node c (*i.e.*, $Sent(c) = \{a, b, c\}$), and b is c's immediate sentinel. Similarly, $Sent(e) = \{a, b, c, e\}$.

For this problem, complete the following sub-tasks:

1. Implement a randomised procedure for generating random rooted directed graphs. As its result, it should return a graph (in a linked representation) and a dedicated root node.
2. In your report, describe the full sentinel tree for the graph from Figure 1.

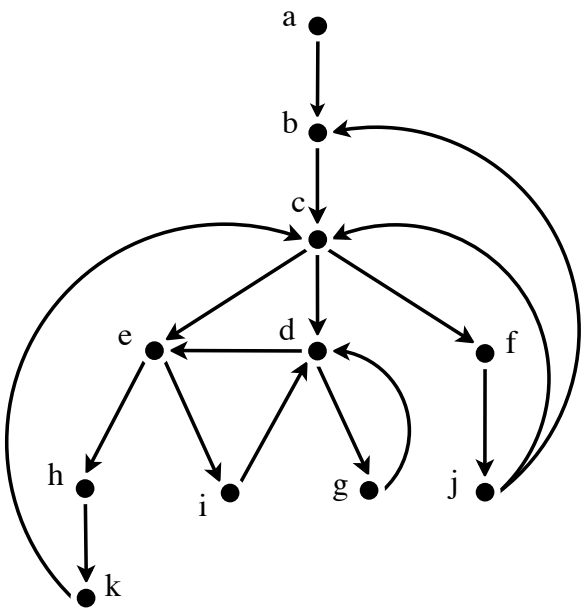


Figure 1: An example graph. The node a is a root.

3. Implement an algorithm for finding a sentinel tree for a given rooted graph.

   Hint: You may represent the tree via a hash-table using the "predecessor" relation, as in the implementation of Dijkstra's algorithm from the lectures.

   Hint: Your algorithm may first construct, for each node, an (unordered) set of its sentinels in the graph, and then choose the immediate one.

   Hint: Consider an algorithm that has a while-loop, which iteratively "refines" the over-approximation of the set *Sent* of sentinels for each node (by removing non-sentinels from it), starting from the initial over-approximation: $Sent(v_0) = \{v_0\}$ and $Sent(v) = V$ for any other $v \neq v_0$. It is up to you to figure out the condition for the loop and what instructions should be in its body.

   Hint: The set of a node v's sentinels is intimately related to the sentinels of v's predecessors.

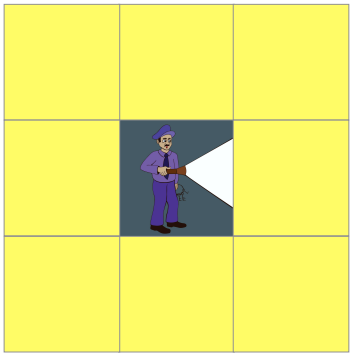
4. Test your algorithm on the graph from Figure 1 and also via the graph generating-procedure designed above, by constructing random paths in a rooted directed graph and checking the sentinel property.

5. Argue that your algorithm terminates for any graph (*i.e.*, state its loop *variant*).

6. Estimate the complexity of your algorithm in terms of $|V|$ and $|E|$.

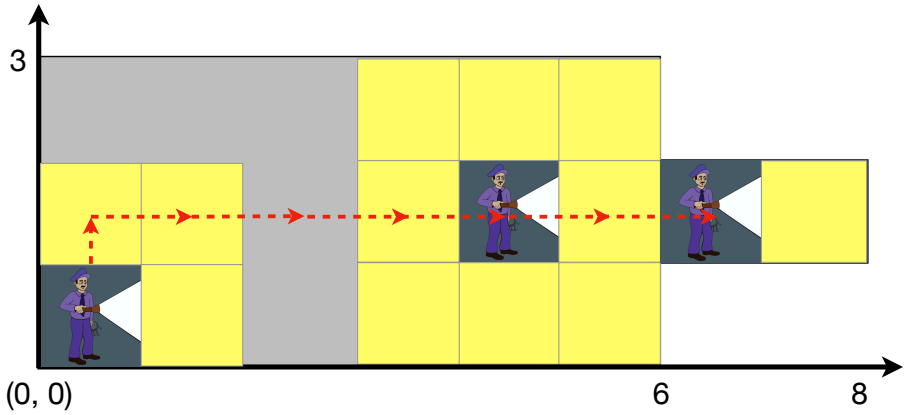
1.3 A Watchman in the Dark

Workload: Two persons



In this task, which can be completed by two students working jointly, you will be solving the problem of creating a route for a lonely watchman that guards a very dark room, being armed only with a feeble torch. A room is represented by a two-dimensional rectilinear polygon with all coordinates being integer values. The watchman occupies one square 1×1, and his position is considered to be the bottom left of this square. The watchman's torch provides enough light to cover the eight squares adjacent the current watchman's position. Following the laws of physics, the light cannot spread through the walls or go around the corners.

Your goal in this task is to compute for a watchman, who starts his shift at a position $(0, 0)$, the best possible *route* to walk through the dark room, while illuminating all of its area. For example, consider a room defined as the polygon with coordinates `[(0, 0); (6, 0); (6, 1); (8, 1); (8, 2); (6, 2); (6, 3); (0, 3)]` and shown on the image below:



In order to check the entire room, the watchman, positioned initially in the coordinate $(0, 0)$ can move by following the route defined by the list of coordinates `[(0, 0); (0, 1); (1, 1); (2, 1); (3, 1); (4, 1); (5, 1); (6, 1)]`. This is a valid route, as each next position is adjacent to the previous one. There is also no need for the watchman to step to any other squares, as the light of the watchman's torch can illuminate the remaining parts, as he is walking. The figure above shows an initial $(0, 0)$, some intermediate $(4, 1)$, and the final $(5, 1)$ positions of the watchman following this route.

Your solution to the watchman problem should contain these components:

1. A function that takes a rectilinear polygon defined by a list of integer-valued 2-dimensional coordinates of its nodes, and

   - checks that the polygon contains the 1×1 square positioned at coordinate $(0, 0)$,
   - computes a valid route for the 1-square watchman carrying a torch, which illuminates a 3×3 square around him, so by following this route the watchman could illuminate the entire room.

   A watchman's route is *valid*, if:

   - it starts at the initial watchman's position, *i.e.*, $(0, 0)$;
   - it is connected, *i.e.*, each next square is reachable from the previous one (*connectivity*);
   - it does not force the watchman to walk through the walls or outside of the room (*no collisions*);
   - by following it and casting the light of the watchman's torch at each location, the entire room is illuminated by the end (*full room coverage*).

   The route in the example above is valid, as it satisfies all these conditions.

2. A suite of tests for validating your solutions on some randomly generated (well-formed) rooms.

3. A file with valid solutions for the ten specific rooms available in the supplementary file.[1]

   In your submitted file, each solution must be represented by a separate line, formatted as `n: (0, 0); (x1, y1); ...`, where `n` is a number of the corresponding room, followed by a colon and a semicolon-separated list of the coordinates of the route. That is, the format is similar to encoding of the rooms in the provided file.

4. A high-level explanation of your algorithm, as well as its optimisations, in your report.

In your implementation, feel free to use any libraries from the lectures. While it might be difficult to find the most optimal (*i.e.*, the shortest) route, please, do your best to come up with a procedure that finds a "reasonably" good solution, *i.e.*, *does not* make the watchman to step into every single square of the room, but relies on the range of his torch instead. That said, even the best solution might require amount of back-tracking (which is allowed), forcing the watchman to step on the same square more than once. While your procedure is allowed to be computationally expensive (explain its complexity in the report), it should terminate in a reasonable time (within 20 seconds) for the ten rooms from the set above.

There will be an informal competition amongst the teams taking on this task, with the ranks allocated according to the overal better results for the ten special rooms from the file above (assuming those solutions were obtained algorithmically). The winners will be announced on Canvas and will be granted the right to openly brag about it. No additional points will be awarded for winning the competition.

5. Implement a visualisation for a solution of the watchman problem. Given a room, compute a watchman's route for it, and then show his movement, colouring the illuminated parts of the room as he walks. Use delays (as in the code from the lectures) to implement animation. Make sure that the light area is rendered correctly (*e.g.*, the light does not spread beyond the walls).

   Hint: Consider scaling your rooms for better visibility in the visualisation.

6. Implement a "video game" to solve the watchman's problem. The player can use keyboard keys/arrows to move the watchman, so their movements are recorded. Once all of the area is illuminated, the game ends with the "score" equal to the length of the produced route. Use the results of your game for testing your algorithmic solutions.

2 Report

Maximal grade for this part: 15 points

As in the case of midterm, the reports are produced and submitted individually, by each student. The report should cover the following aspects of your solutions:

1. Which task from Section 1 you were responsible for (perhaps, jointly with another member of your team in the case of Problem 1.3).

2. A high-level but rigorous description of your solution for the task you were responsible for.

3. Descriptions of solutions for other tasks, submitted by your team members (in the case if they contributed to the submitted codebase). It is allowed to use the visualisation tools developed by your team members, as long as the pictures were produced by yourself independently (possibly with a help of the main implementers), and come with your commentary on what is being demonstrated.

4. Optional assignments of peer-awarded points to your team members.

The points for the individual reports will be awarded on the basis of clarity of the explanations. A simple way to increase the clarity of technical writing is to choose illustrative examples. Paraphrasing the code verbatim is typically a bad idea. Make sure that, even for the tasks you were not responsible fore, the examples in your report are your own. Finally, try to be concise. As much as it is entertaining to read about your process of discovering solutions, I have a limited amount of time to do the grading.

[1] https://ilyasergey.net/YSC2229/resources/2019/rooms.txt