COMP 2002 – Assignment 3 (6%)

Due: 11:59pm, Tuesday, Nov. 14, 2023

Learning Objectives

The goal of this assignment is to gain experience with hash functions and binary search trees.

Instructions

Hash functions are composed of hash codes and compression functions. Hash codes take an input and converts it to an integer. The compression function takes that integer and reduces it to a smaller range. A key component of a good hash function is that it minimizes collisions, occurring when a hash function evenly spreads inputs amongst the possible range of integers.

Here are some useful built-in Python functions that will be helpful for this assignment.

- `bin(x)` – returns the binary representation (as a string) of the integer x.
- `int(x, 2)` – returns the integer represented by a binary string representation, x.
- `ord(ch)` – returns the integer corresponding to the single Unicode character, ch.
- `chr(x)` – returns the string representing the character given by the Unicode integer, x.
- `x « d` – shift the bits of an integer, x, to the left d spaces.
- `x » d` – shift the bits of an integer, x, to the right d spaces.
- `x ^ y` – returns the "bitwise exclusive or" (XOR) of x and y.
- `x.strip()` – removes any whitespace or new line characters from the string, x.

Question 1: (40 pts)

You have discovered a secret message. You suspect that it might have something to do with the final exam (a study guide? a bonus question?), but it is unclear since the message is encrypted by a hash function.

The message is a series of integers, representing hash codes for portions of the message. You realize that each integer corresponds to a 32-bit segment of the original message, which in turn represents four 8-bit characters. Each character has been obscured by an "exclusive or" (XOR) operation with some unknown cipher value. The value of the cipher that each character has been XOR'd with is unclear.

The "exclusive or" between two bits is 0 if both bits are 0 or 1, and the "exclusive or" is 1 if one of the bits is 1 and the other 0. That is, it returns 0 if the two bits are the same or 1 if they are different. For example, `01101 ^ 10101 = 11000`.

Importantly, you know that when a value is XOR'd with another value, repeating the XOR again will return the original value. That is, if `a ^ b = c`, you know that `a` can be recovered again by the operation `c ^ b = a`.

The integer cast hash code for each 32-bit segment of the original message is given below. Determine the value of the 8-bit XOR cipher that correctly unhashes each character to recreate the original string and uncover the secret message.

- `3603614414`
- `3448017297`
- `3385444752`
- `3352415178`
- `3420248976`
- `3721515921`
- `3386886877`
- `3598829699`
- `4208118481`
- `2414407563`
- `4173892254`

Submit the cipher value, the secret message that you decoded, and the Python code that you used to find the cipher value for the XOR operation that unhashes and reveals the secret message.

Question 2: (25 pts)

Download the following files:

- `vertexes.csv`
- `lines.csv`
- `map_render.py`
- `binary_search_tree.py`

The data files contain the vertexes and lines representing a map from a video game level. Each line of `vertexes.csv` represents a vertex given by its x and y coordinates. Each line of `lines.csv` represents a line segment connecting two vertexes.

The source file `map_render.py` contains functions to load the data and draw the map. Inside `binary_search_tree.py` is a class, `BinarySearchTree`, that implements a binary search tree using a linked structure.

Adapt the `BinarySearchTree` class to store vertexes.

Modify the `_search()`, `search()` and `insert()` methods of the `BinarySearchTree` class to work with the (x, y) positional data of vertexes instead of (k, v) pairs in each node. Items are inserted into the binary search tree by comparing x-coordinates first, and if they're the same, then comparing y-coordinates. The binary search tree must still use a linked structure.

Question 3: (35 pts)

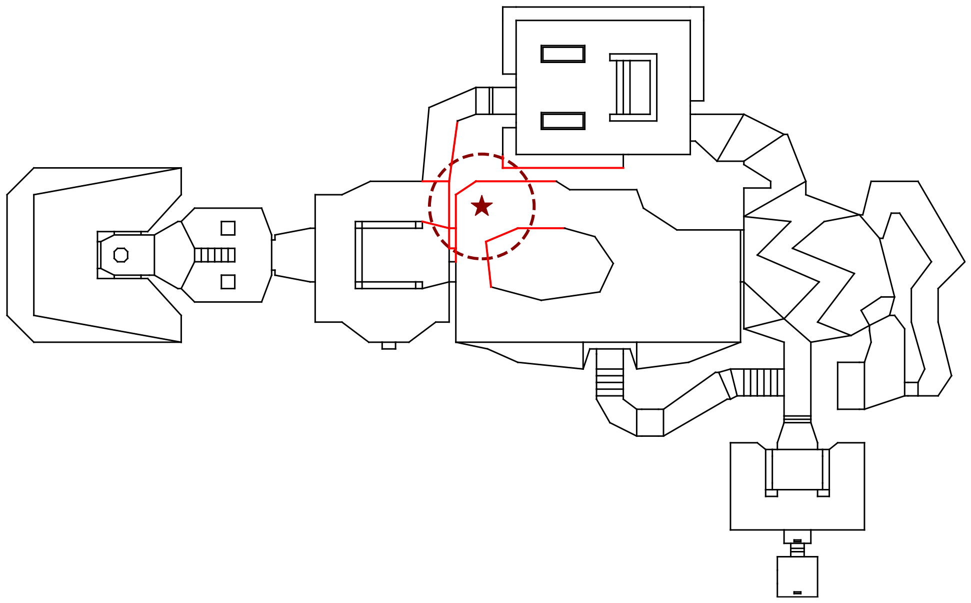
A number of video games implement rendering distances for computational performance. Objects or surfaces past a certain distance are not rendered. Often these games using a 'fogging' technique so items in the distance fade away rather than sharply disappearing.

Implement the `find_nearby_vertexes(x_centre, y_centre, radius)` method in the `BinarySearchTree` class.

This method finds all potential surfaces within proximity of a player anywhere in the map. Given the radius, `radius`, of a circle centred on the point given by `x_centre` and `y_centre`, it returns a list of (x, y) tuple pairs representing the vertexes within the circle.

Do not use recursion to implement this method.

For example, the figure shows a player located at position (1500, -3000). All surfaces that are within a radius of 250 are coloured red. The list of nearby vertexes would be:
(1344, -2880), (1344, -3104), (1376, -3200), (1376, -3104), (1376, -2944), (1376, -2959), (1472, -2880), (1520, -3168), (1600, -2816), (1664, -2816), (1664, -2880), (1672, -3104).



The player is at the position of the star. Within rendering distance is everything inside the red circle. The walls that are within visible range of the player are coloured red.

Submission

Submit a single zip file containing the Python source file for Question 1 and the Python file containing your modifications to the `BinarySearchTree` class for Questions 2 and 3.

Python source code should be `*.py` plain text. The only file types allowed aside from Python source code (`*.py`) are pdfs and plain text (`*.txt`). Do not submit Word documents or rich text format documents. They will not be marked. Only submit a single zip archive. Do not submit files archived in rar format. That may result in your assignment not being graded.

Name all files with the format "firstname_lastname_studentid_...". Make sure to include your name and student ID as comments at the top of all Python source files.

Late submissions will be subject to a 10% penalty for each hour past the deadline.

Attribution

Submissions must represent your independent work.

If your submitted work includes unacknowledged collaboration, code materials, ideas or other elements that are not your original work, it may be considered plagiarism or some other form of cheating under MUN general regulations 6.12.4.2 and academic penalties will be applied accordingly.

The submission of work that has been created by generative artificial intelligence (GAI) tools and presented as a student's original work is considered an academic offence in this course. Using AI tools without proper citation constitutes plagiarism, and your work will be subject to the appropriate Memorial's Academic Misconduct policy.

If your submission contains any contribution by others, including internet sources and classmates, then you should include an attribution section detailing the extent of these contributions. This will also help distinguish what elements of the submission are original. You may not receive full credit if your original elements are insufficient, but can lessen penalties for plagiarism or copying if you acknowledge your sources.

Github

I encourage you to store and version your work on Github. It is good practice to do so as everyone uses git in the real world.

However, **it is a requirement that git repositories containing assignment material be private.** University regulations section 6.12.4.2 consider it cheating if you allow your work to be copied. There will be zero tolerance for this.