Structural formula to naming organic molecules.

Table of Contents

[Analysis 2](#_Toc147820674)

[Introduction 2](#_Toc147820675)

[Initial idea 2](#_Toc147820676)

[Final idea 2](#_Toc147820677)

[Naming conventions 2](#_Toc147820678)

[Background 3](#_Toc147820679)

[Problem definition 3](#_Toc147820680)

[Problem Research 3](#_Toc147820681)

[Interview 3](#_Toc147820682)

[Functional groups 3](#_Toc147820683)

[Mandatory functional groups 4](#_Toc147820684)

[Additional functional groups 4](#_Toc147820685)

[Research into similar applications 5](#_Toc147820686)

[Marvin JS 5](#_Toc147820687)

[Modelling of the problem and solution 6](#_Toc147820688)

[Challenges to overcome. 6](#_Toc147820689)

[Data volume 7](#_Toc147820690)

[Objectives 7](#_Toc147820691)

[Must 7](#_Toc147820692)

[Could 8](#_Toc147820693)

[wont 8](#_Toc147820694)

[Design 8](#_Toc147820695)

[Molecule 8](#_Toc147820696)

[Adjacency matrices creation 8](#_Toc147820697)

[representation 8](#_Toc147820698)

[Longest carbon chain 9](#_Toc147820699)

# Analysis

## Introduction

### Initial idea

#### Initially I wanted to create a chemistry revision helper program, that could test students’ knowledge of organic molecules and track their progress. To generate test questions this would rely upon a random organic molecule generator, to build different classes and complexity of organic chemistry.

#### Issues I found.

* Each of the tasks would have been their own NEA project.
  + The random generator
    - Would require very complex code.
    - Uses adjacency matrices.
    - There isn’t much to write an analysis about.
    - There isn’t much other code to use to bulk out the project.
  + Progress checker
    - Very low-level coding
    - Takes a long time to code it all.
    - Not much that can be graded.

### Final idea

The final idea that I landed on was an organic molecule, Namer. This would be done on a command line as it would require an input of a sequence of C ,H , and () then the name for the molecule will be given following the IUPAC rules

Pros

* Not too difficult code to go into it.
* A lot of room to grow the project if needed.
* I can do this in C# and don’t need to learn a new language.

Cons

* Could be difficult to add complex code required.
* The more complicated molecules will require more adjacency matrices and would present problems.
* Could be difficult to write detail around detail added as most of it is hidden.

### Naming conventions

Naming conventions refer to the universal way of naming organic molecules so it can be recognised throughout the world without much of a language barrier separating these standards and preventing the scientific development of the world. Which means if a molecule is misnamed the whole experiment could be nulled especially if dealing with larger and expensive equipment that costs thousands of pounds per run.

For this project I will be using IUPAC nomenclature the rules can be found here [https://www.angelo.edu/faculty/kboudrea/organic/IUPAC\_Handout.pdf](https://www.angelo.edu/faculty/kboudrea/organic/IUPAC_Handout.pdf%20)

### Background

Many chemistry students of all educational levels find it annoying to have to name molecules and it is easy to mess up. And it can distract a researcher from key problems or aspects of an experiment that cause a health risk if they miss a functional group or misidentify it.

### Problem definition

The project is used to solve the problem of naming organic molecules in their scientific names along with some extra information about it.

# Problem Research

This problem will require some advanced programming techniques. This means that I will need to do some research into how to use adjacency tables. I shouldn’t need to make this into a website at first, so I won’t need to learn too much html and JavaScript. Finding the longest chain can relate to the travelling salesman problem.

## Traveling Salesman Problem

The TSP is one of the most intensively researched problem in mathematics. The TSP talks about a situation where a salesman must visit n cities without revisiting cities. To then end up back at the first city. At first this problem seems easy as it can be solved via brute force. However, it will require you to test every possible combination ((n-1)!). this can be related to my problem by having to compare every combination of paths.

## Limitations

Because of the brute force attempt to the problem this project can only be solved in polynomial time in reference to the number of different paths.

## Interview

## Functional groups

These functional groups are all part of the A-level chemistry spec however I don’t think I will be able to complete all the groups in my code so will list the mandatory ones and the additional ones can be added to beef up the project if I have the timeA screenshot of a cell phone

Description automatically generated

R = further carbon chain

### Mandatory functional groups

* Alkane
* Alkyl groups

### Additional functional groups

* Ester
* Primary amine
* Nitrile
* Cyclic
* Carboxylic acid
* Ketone
* Aldehyde
* Alcohol
* Alkenes

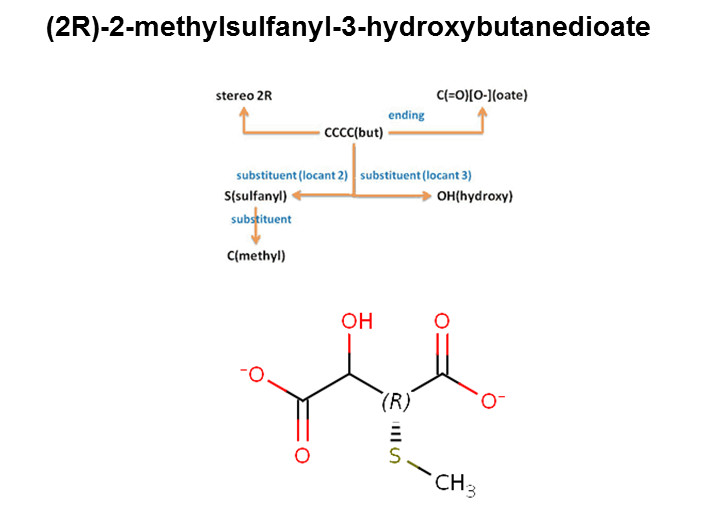
The reason I am only doing alkanes and alkyl groups is the other functional groups will require more adjacency matrices counting the bonds of c-h rather than just c-c. most of the other functional groups require more elements further extending the complexity of the rules and adjacency matrices.

## Research into similar applications

### Marvin JS

One application I found was [Marvin JS](https://chemaxon.com/chemical-naming-and-structure-conversion) (Marvin, 2023) their identification and naming process goes through a large database of very many molecules. This seems to be a bit too inaccessible to me and doesn’t follow the pathway I am wanting to follow to name the molecule. Marvin JS also does its naming convention based off a drawing input on a premade molecule drawing application. Which I was planning to use for my previous idea, but I deemed the project around this to be too much for one project.

Although I’m not going to use Marvin JS’s application, they do have a rather useful flow table for molecule identification.



This model shows the process to naming a compound. It starts by finding the longest carbon chain. Then finds the main functional group on the end of the chain then looks at the other functional groups (OH, S, CH3) it then marks down the location on the carbon chain that it is connected to or the connection to another functional group (this last bit may be a bit too complicated for the NEA)

## Modelling of the problem and solution

A molecule structure with letters and numbers

Description automatically generatedCH3CH(CH3)CH3
CHHHCH(CHHH)CHHH

C1
C2
C3
C4
H1
H2
H3
H4
H5
H6
H7
H8
H9
H10
C1
0
1


1
1
1







C2
1
0
1
1



1






C3

1
0





1
1
1



C4

1

0







1
1
1
H1
1



0









H2
1




0








H3
1





0







H4

1





0






H5


1





0





H6


1






0




H7


1







0



H8



1







0


H9



1








0

H10



1









0
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
Ink Drawings
= longest chain calculator
Validity checker = counting the number
                                 of 1s connected to 
each atom


Set rules.

Carbon only has 4 adjacencies.

Hydrogen only has 1.

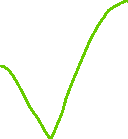
Oxygen has 2.

## Challenges to overcome.

1. Finding the longest carbon chain then number each carbon.
2. Use the right chain based on the functional groups present. (only if I were to expand into further functional groups)// only necessary if I expand out of alkanes and methyl groups

A picture containing diagram, line, design, origami

Description automatically generated



1. Validating the molecule to make sure each element has the correct number of attachments.
2. Differentiating between cyclic functional groups

## Data volume

My project has no log in system. Therefore, there will be no need for a large database for all users who have logged in. However, I plan to create a database for all molecules named to help prevent long load times. This could mean the data base will expand with use roughly linearly.

I don’t intend to make this open source so I wont have to have a server to deal with client packages

## Objectives

### Must

* Be able to receive inputs of structural formula.
  + Be able to convert it into a carbon and bracket only string.
* Identify longest carbon chain.
  + Via using adjacency matrices
  + connect the respective carbons (arcs)
  + Able to work around/with “(“and”)”
* Find functional groups location on carbon chain.
  + Number the final carbon chain to find where the extra alkyl groups attach.
    - Recursively check arcs for the carbon to find alternative routes to count as alkyl groups.
    - Find the longest chain from the root node that doesn’t follow the original longest chain.
    - This must be repeated in case you get alkyl groups on alkyl groups.
* Output the correct name for the molecule stated.
  + This should be based on the IUPAC nomenclature.

(https://www.angelo.edu/faculty/kboudrea/organic/IUPAC\_Handout.pdf)

* Create a Database
  + stores all the molecules that have previously been named with their structural formula.
  + check the database for molecule before it names it as a new molecule.

### Could

* Identify:
  + Alkenes (C=C)
  + Aldehydes (R-C=O-H)
  + Ketones (R-C(=O)-R)
  + Carboxylic acid (R-COOH)
* Give the MR of the molecule.
  + By having a dictionary of elements and their Relative Atomic Mass
* Give the ways the molecule can react.

### wont

* Add a drawing GUI.
* Get an image of the molecule named.

# Design

## Molecule

### Adjacency matrices creation

I took a skeleton code from stack overflow (<https://stackoverflow.com/questions/15306040/generate-an-adjacency-matrix-for-a-weighted-graph>)

And I modified the classes more efficiently in more organised classes. I will also adjust the node creations to be dynamic to an input based on the number of “C” in the string. I also adjusted the arc creation to follow rules to correctly create arcs based on a string input.

### representation

The molecule will be inputted, and the software will need to rewrite it in a way it can understand. For this project I will need to convert the structural formula to what I call extended structural formula then to an adjacency table as shown in the table below eg CH3CH(CH3)CH3

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | C1 | C2 | C3 | C4 | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 | H10 |
| C1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C2 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| C3 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| C4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| H1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H5 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H6 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H7 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H8 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H9 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H10 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Issues with this methos is it makes it difficult to validate the molecule that have double bonds in them.

### Longest carbon chain

This process is rather simple. Or so I thought, as of now I have spent around 6 lessons purely working on these problems using graphs, adjacency matrices, search algorithms etcetera. And now I finally think I have 2 possible solutions to the problem. Initially when I started this problem, I thought an adjacency matrix and a 20-line code would solve my problem.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | C1 | C2 | C3 | C4 | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 | H10 |
| C1 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| C2 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| C3 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| C4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| H1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H2 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H3 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H4 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H5 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H6 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H7 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H8 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H9 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| H10 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Above shows my first idea.

Starting at c2,c1 with the chain length of 2 the checker would move to the cell to the right and check if it has a “0”if it does then it would check the cell below it and if has a one then it would check the “1” on its right and if there was a “0” it would go down unless it is the second “0” the code would stop and out put the number of times the program moved one cell. This initially looked like it was good however it didn’t consider the possibility on methane-based molecules with one carbon. It also didn’t allow for the first carbon to not be the first in the chain.

And to fix the positioning problem I used Butane and started at the second carbon this was the adjacency matrix that it produces.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | C1 | C2 | C3 | C4 | H1 | H2 | H3 | H4 | H5 | H6 | H7 | H8 | H9 | H10 |
| C1 |  | 1 |  | 1 | 1 | 1 |  |  |  |  |  |  |  |  |
| C2 | 1 |  | 1 |  |  |  | 1 | 1 |  |  |  |  |  |  |
| C3 |  | 1 |  |  |  |  |  |  | 1 | 1 | 1 |  |  |  |
| C4 | 1 |  |  |  |  |  |  |  |  |  |  | 1 | 1 | 1 |
| H1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| H2 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| H3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| H4 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| H5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| H6 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| H7 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| H8 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| H9 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| H10 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |

First program second idea

If I had followed the initial algorithm would have outputted a chain length of 3 which is the wrong output

My second idea was to check the cell above and below and follow the first one. This would have outputted the correct output of 4 however this didn’t consider alkyl groups, so you could ask “why don’t you check both the lines from carbons attached to c1”. If you asked this, then you would be wrong as it assumes C1 lies on the longest path.

At this point I started to look at graphs and traversal techniques.

Now I have decided on a solution to the problem. I will set a root carbon. Then I would make a list of all the different routes that could be taken without going back on itself. Then I would compare lists and I would find the longest chain and then compare it to the second longest and find the similar carbons then add to the list of it makes the chain longer from the point of the last common carbon. This will repeat until all the combinations have been tried then an output will be given.

A diagram of numbers and a diagram of numbers

Description automatically generated

This will extend with more complicated molecules. It will iterate and compare every combination of paths. And record the length of each combination. At the end the list with the longest length will be the longest path. This iteration makes the solution of this problem go into polynomial time in a similar fashion to the traveling salesman problem.