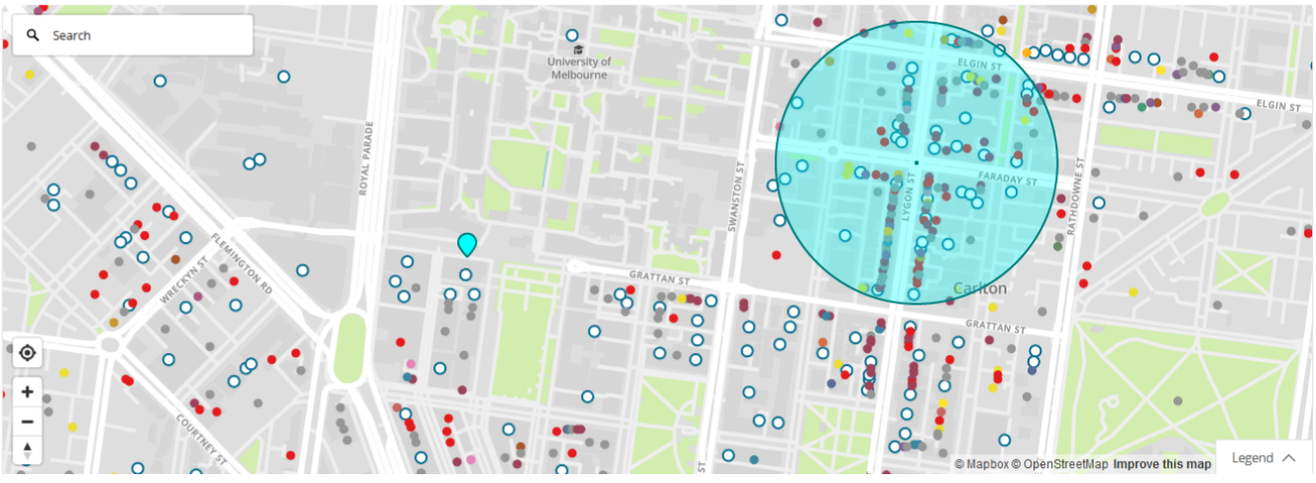


COMP20003 Algorithms and Data Structures
Second (Spring) Semester 2020
[Assignment 2]
Melbourne Census Dataset
Implementing Map Functions with K-D Trees

Handed out: Monday, 7 of September
Due: 11:59 PM, Sunday, 20 of September
Marks: 15 (15% of total mark)

Purpose

The purpose of this assignment is for you to:

- Increase your proficiency in C programming, your dexterity with dynamic memory allocation and your understanding of more advanced data structures (K-D trees).
- Increase your understanding of how computational complexity can affect the performance of an algorithm by conducting orderly experiments with your program and comparing the results of your experimentation with theory.
- Increase your proficiency in using UNIX utilities.

Background

Interactive navigation tools like Google Maps and GPS navigation systems are commonplace, but how do these systems actually work? Spatial datasets can be very large: `OpenStreetMap`, an open-source global mapping dataset, contains over 10 million points of interest. Various algorithms and data structures have been developed to allow users to quickly search and navigate these large spatial datasets.

Your task

In this assignment, you will create a K-D tree to support interactive map functionality for the City of Melbourne Census of Land Use and Employment (CLUE) dataset. A user will be able to query locations to find nearby businesses.

In Assignment 1, you wrote code to read the census data from a file, insert the records as nodes in a linked list, and allow users to search for records by trading name. In this assignment, you should modify your code to insert records into a K-D tree and allow users to search by x,y coordinates. You can use your own Assignment 1 code for this assignment or the sample solution we have provided.

Dataset

This assignment uses the same dataset as Assignment 1, which is a subset of the Business Establishment Trading Name and Industry Classification 2018 dataset, accessed from:
`https://data.melbourne.vic.gov.au/Business/Business-establishment-trading-name-and-industry-c/vesm-c7r2`

The `<x coordinate>` and `<y coordinate>` columns should be used as the 2-D key to store and query records. The other columns can be treated as the associated `<data>`.

Deliverable 1 - Source code

Stage 1 - What's here? (7 marks)

In stage 1, you will implement the basic functionality for an interactive map that allows a user to click on locations and retrieve data about the nearest point of interest. Instead of clicks, your code will accept (x,y) pairs from `stdin`, find the closest business establishment to that location in the dataset, and output the information about that establishment to a file.

Your `Makefile` should produce an executable program called `map1`. This program should take two command line arguments: (1) the name of the data file used to build the tree, and (2) the name of an output file.

Your `map1` program should:

- Construct a K-D tree to store the information contained in the data file specified in the command line argument. Each record (row) should be stored in a separate Node.
- Handle duplicates (businesses located at exactly the same x,y coordinates) by chaining them together in a linked list connected to a single Node in the K-D tree. Exact duplicate locations should not be added as separate Nodes in the tree.
- Accept locations queries from `stdin` and search the tree for the location nearest to the query point. The record(s) of the business(s) at this location should be printed to the output file. If there are multiple businesses at this location, all of them must be included in the output.
- In addition to outputting the record(s) to the output file, the number of key comparisons performed during the search should be written to `stdout`.

For testing, it may be convenient to create a file of keys to be searched, one per line, and redirect the input from this file. Use the `UNIX` operator `<` to redirect input from a file.

Example input

- `map1 datafile outputfile` then type in queries; or
- `map1 datafile outputfile < queryfile`

Queries should be entered as x,y pairs separated by a space: `x y`

Example output

This is an example of what might be output to the file after two queries:

```
144.959522 -37.800095 --> Census year: 2018 || Block ID: 240 || Property ID: 104466 || Base property ID: 104466 || CLUE small area: Carlton || Trading Name: The Bio 21 Cluster || Industry (ANZSIC4) code: 6910 || Industry (ANZSIC4) description: Scientific Research Services || x coordinate: 144.9593 || y coordinate: -37.8002 || Location: (-37.80023252, 144.9592806) ||
144.959522 -37.800095 --> Census year: 2018 || Block ID: 240 || Property ID: 104466 || Base property ID: 104466 || CLUE small area: Carlton || Trading Name: The University of Melbourne || Industry (ANZSIC4) code: 8102 || Industry (ANZSIC4) description: Higher Education || x coordinate: 144.9593 || y coordinate: -37.8002 || Location: (-37.80023252, 144.9592806) ||
144.959522 -37.800095 --> Census year: 2018 || Block ID: 240 || Property ID: 104466 || Base property ID: 104466 || CLUE small area: Carlton || Trading Name: The Co-Op Bookstore || Industry (ANZSIC4) code: 4244 || Industry (ANZSIC4) description: Newspaper and Book Retailing || x coordinate: 144.9593 || y coordinate: -37.8002 || Location: (-37.80023252, 144.9592806) ||
144.959522 -37.800095 --> Census year: 2018 || Block ID: 240 || Property ID: 104466 || Base property ID: 104466 || CLUE small area: Carlton || Trading Name: Baretto Cafe || Industry (ANZSIC4) code: 4511 || Industry (ANZSIC4) description: Cafes and Restaurants || x coordinate: 144.9593 || y coordinate: -37.8002 || Location: (-37.80023252, 144.9592806) ||

0 0 --> Census year: 2018 || Block ID: 571 || Property ID: 602254 || Base property ID: 602254 || CLUE small area: Kensington || Trading Name: Shine Australia || Industry (ANZSIC4) code: 5511 || Industry (ANZSIC4) description: Motion Picture and Video Production || x coordinate: 144.9081 || y coordinate: -37.7851 || Location: (-37.78505447, 144.9081466) ||
```

This is an example of what might be output to `stdout`:

```
144.967058 -37.817313 0.0005 --> 4815
144.963089 -37.799092 0.0005 --> 359
```

Note that the key is output to both the file and to `stdout`, for identification purposes. Also note that the number of comparisons is only output at the end of the search, so there is only one number for key comparisons per key, even when multiple records have been located for that key.

The format need not be exactly as above; variations in whitespace/tabs are permitted. *The number of comparisons above has been made up, do not take it as an example of a correct execution!*

Stage 2 - Radius search (2 marks)

In stage 2, you will code a function that allows the user to find all of the business establishments within some distance of a query point. Your code will accept (x,y,radius) triplets from `stdin`, find all business establishments within the requested radius of the x,y point, and output the information about those establishments to a file.

Your `Makefile` should produce an executable program called `map2`. This program should take two command line arguments: (1) the name of the data file used to build the tree, and (2) the name of an output file.

Your `map2` program should:

- Construct a K-D tree to store the information contained in the data file specified in the command line argument, exactly as in Stage 1. Note that you can (and should!) reuse your code from Stage 1 to do this step.

- Accept x,y,radius queries from `stdin` and search the tree for all locations within the requested radius of the x,y point. These records should be printed to the output file. When there are multiple businesses at the same location, all of these records should be included in the output.
- If no business establishments are located with the query radius, your code must output the word `NOTFOUND`.
- In addition to outputting the above data to the output file, the number of key comparisons performed during the search should be written to `stdout`.

Example input

- `map2 datafile outputfile` then type in queries; or
- `map2 datafile outputfile < queryfile`

Queries should be entered as x,y,radius triplets separated by spaces: `x y r`

Example output

This is an example of what might be output to the file after two queries:

```
144.967058 -37.817313 0.0005 --> Census year: 2018 || Block ID: 15 || Property ID: 109260
|| Base property ID: 109260 || CLUE small area: Melbourne (CBD) || Trading Name: Hungry Jack's Pty Ltd
|| Industry (ANZSIC4) code: 4511 || Industry (ANZSIC4) description: Cafes and Restaurants || x coordinate:
144.9668 || y coordinate: -37.8171 || Location: (-37.81711586, 144.9668418) ||
144.967058 -37.817313 0.0005 --> Census year: 2018 || Block ID: 15 || Property ID: 109258
|| Base property ID: 109258 || CLUE small area: Melbourne (CBD) || Trading Name: McDonalds || Industry
(ANZSIC4) code: 4511 || Industry (ANZSIC4) description: Cafes and Restaurants || x coordinate: 144.9669
|| y coordinate: -37.8172 || Location: (-37.81724484, 144.9669126) ||
144.967058 -37.817313 0.0005 --> Census year: 2018 || Block ID: 15 || Property ID: 104015
|| Base property ID: 104015 || CLUE small area: Melbourne (CBD) || Trading Name: Dangerfield || Industry
(ANZSIC4) code: 4251 || Industry (ANZSIC4) description: Clothing Retailing || x coordinate: 144.9668
|| y coordinate: -37.8174 || Location: (-37.81741866, 144.9668092) ||
144.967058 -37.817313 0.0005 --> Census year: 2018 || Block ID: 15 || Property ID: 109257
|| Base property ID: 109257 || CLUE small area: Melbourne (CBD) || Trading Name: Young & Jacksons ||
Industry (ANZSIC4) code: 4520 || Industry (ANZSIC4) description: Pubs, Taverns and Bars || x coordinate:
144.967 || y coordinate: -37.8174 || Location: (-37.81735593, 144.967023) ||
144.967058 -37.817313 0.0005 --> Census year: 2018 || Block ID: 15 || Property ID: 109259
|| Base property ID: 109259 || CLUE small area: Melbourne (CBD) || Trading Name: Souvenir Collection
|| Industry (ANZSIC4) code: 4310 || Industry (ANZSIC4) description: Non-Store Retailing || x coordinate:
144.9669 || y coordinate: -37.8172 || Location: (-37.8171602, 144.9668989) ||

144.963089 -37.799092 0.0005 --> NOTFOUND
```

This is an example of what might be output to `stdout`:

```
144.967058 -37.817313 0.0005 --> 678
144.963089 -37.799092 0.0005 --> 1356
```

Note that the key is output to both the file and to `stdout`, for identification purposes. Also note that the number of comparisons is only output at the end of the search, so there is only one number for key comparisons per key, even when multiple records have been located for that key.

The format need not be exactly as above; variations in whitespace/tabs are permitted. *The number of comparisons above has been made up, do not take it as an example of a correct execution!*

Requirements

In each stage, the following implementation requirements must be adhered to:

- You *must* write your implementation in the C programming language.
- You *must* write your code in a modular way, so that your implementation could be used in another program without extensive rewriting or copying. This means that the search tree operations are kept together in a separate .c file, with its own header (.h) file, separate from the main program.
- Your code should be easily extensible to different data structuress. This means that the functions for insertion and search take as arguments not only the item being inserted or a key for searching, *but also a pointer to a particular tree*, e.g. `insert(tree, item)`.
- As in Assignment 1, you should include a `Makefile` to compile your code.

Programming Style (2 Marks)

Programming style will be assessed as in Assignment 1 and is worth 2 marks.

Deliverable 2 - Experimentation (4 marks)

You will run various files through your program to test its accuracy and also to examine the number of key comparisons used when searching. We have provided a few different versions of the .csv file that can be used to build the tree. You should test your code with a variety of inputs and report on the number of key comparisons used by your code in Stage 1 and Stage 2. You will compare these results with each other and, importantly with what you expected based on theory *(big-O)*.

Your experimentation should be systematic, varying the size and characteristics of the files you use (e.g. sorted or random), and observing how the number of key comparisons varies. Repeating a test case with different keys and taking the average can be useful.

Some useful UNIX commands for creating test files with different characteristics include `sort, sort -R` (`man sort` for more information on the `-R option`), and `shuf`. You can randomize your input data and pick the first x keys as the lookup keywords.

If you use only keyboard input for searches, it is unlikely that you will be able to generate enough data to analyze your results. You should familiarize yourself with the powerful `UNIX` facilities for redirecting standard input (`stdin`) and standard output (`stdout`). You might also find it useful to familiarize yourself with `UNIX` pipes `'|'` and possibly also the `UNIX` program `awk` for processing structured output. For example, if you pipe your output into `echo ''abc:def'' | awk -F ':' '{print $1}'`, you will output only the first column (`abc`). In the example, `-F` specifies the delimiter. Instead of using `echo` you can use `cat filename.csv | awk -F ';' '{print $1}'` which will print only the first column of the filename.csv file. You can build up a file of numbers of key comparisons using the shell append operator `>>`, e.x. `your_command >> file_to_append_to`.

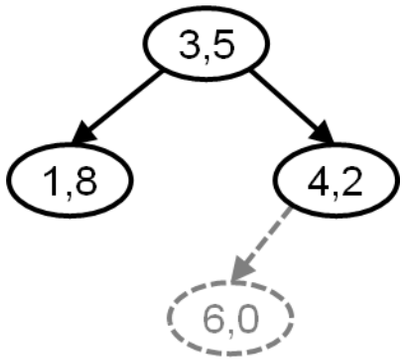
You will write up your findings and submit this report separately from your code. You should present your findings clearly, in light of what you know about the data structures used in your programs and their known computational complexity. Tables and/or graphs are recommended for presenting numeric results. You may find that your results are what you expected, based on theory. Alternatively, you may find your results do not agree with theory. In either case, you should state what you expected from the theory, and if there is a discrepancy you should suggest possible reasons.

A guide on report writing (and a guideline to what we'll expect to see in the report) can be found at: `https://students.unimelb.edu.au/academic-skills/explore-our-resources/report-writing/research-reports`
You are not constrained to any particular structure in this report, but a useful way to present your findings might be:

- Introduction: Summary of data structures and inputs.
- Stage 1 and Stage 2:
  - Data (number of key comparisons)
  - Comparison of the two stages
  - Comparison with theory
- Discussion

Notes on K-D Trees

K-D trees are an extension of binary search trees for k-dimensional keys. When searching, each layer of the tree checks a different dimension of the key. In the 2-D case, the root of the tree should consider only the first element of the key (x) and split left or right depending on whether the first element of the search key is less than or greater than the first element of the root's key. The second layer of nodes should consider only the second element of the key (y). The third layer should consider the first element, etc. To illustrate, consider how the key $(6, 0)$ would be inserted into the K-D tree shown below:



The root node compares the first elements of the keys and since $6 > 3$, it directs the search right. The next node compares the second elements of the keys and since $0 < 2$, it directs the search left. The new node $(6, 0)$ is then inserted as a left child.

Note that it is possible for a key to match an existing node on either x or y but not both. These partial matches are not duplicates and should be inserted as new nodes in the tree. For example, suppose the next node inserted was $(2, 8)$. This would go left from the root because $2 < 3$ and be compared to $(1, 8)$. The second element of the keys match: $8 = 8$, but the keys are not exactly the same $(2, 8) \neq (1, 8)$. So $(2, 8)$ could be inserted as either a left or right child of $(1, 8)$ (for consistency with our testing code, please put equal value keys to the right).

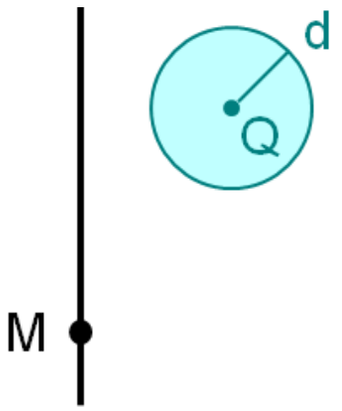
Searching for the nearest point to a query

To find the nearest node to a query point, you will need to search through the tree, keeping track of the closest match found so far, and the squared distance d to that closest match. Start at the root of the K-D tree. At each node, check the distance between the current node and the query location $D = \sqrt{(node_x - query_x)^2 + (node_y - query_y)^2}$. If this distance is lower than the current d, replace the

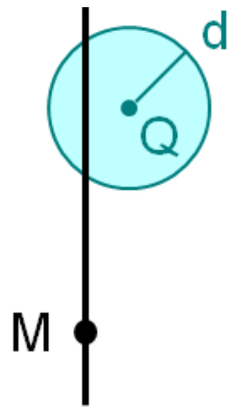
current "closest" match with the current node and set $d = D$. Then compare the current node's key to the query. (As in insertion, only one element of the node's key should be compared to the query – for example, at the root node, compare x values only, and in the first layer nodes compare y values only.) Proceed down one or both branches of the tree depending on the distance between the current node's key and the query:

- If the current node's key is $> d$ from the query, proceed down either the left or right branch of the tree depending on whether the query is less or greater than the current node's key.
- If the current node's key is $\leq d$ from the query, proceed down *both* branches of the tree.

These two cases are illustrated in the images below. In this example, M is a node of the K-D tree which splits the data along the x dimension (indicated by the black line) and Q is the query location. d is the distance to the current "closest" match. In the first case (a), the x distance between M and Q is larger than d, so no points left of M could be closer to Q than the current "closest" match. There is no need to search the left children of M. In the second case, the x distance between M and Q is less than d, so there is a chance that there could be a point left of M that is closer than the current match. In order to guarantee that we find the closest possible point to Q, it is necessary to search both the left and right children of M.



(a) Distance between M and Q is greater than d



(b) Distance between M and Q is less than d

Searching for points in a radius

Searching for points within a radius of a query is similar to searching for the nearest point. Start at the root of the K-D tree, and at each node, check the distance between the current node and the query location. If this distance is within the requested radius, output the information related to this node and proceed down both branches to look for further matches. If the current node is outside the requested radius, check the distance between the current node's key and the query coordinates as above, and proceed down the left, right, or both branches of the tree accordingly.

Conventions and recommendations

For easier testing and debugging, we ask that you follow these conventions:

- The root of the tree should check the x coordinate of the key.
- Equal keys (meaning, keys that match the current node on the portion of the key it checks, but differ from the current node on the other value) should be grouped with the keys greater than the current node, so each node splits keys into values $<$ and $\geq$ the node's key.

The x and y coordinates should be stored as `double` type.

Math functions like `sqrt()` and `pow()` can be found in the `<math.h>` library.

Until you are confident your code is working, you might want to test the radius search function with small radius values (e.g., around 0.0005).

Additional Support

Your tutors will be available to help with your assignment during the scheduled workshop times. Questions related to the assignment may be posted on the Piazza forum, using the folder tag *assignment1* for new posts. You should feel free to answer other students' questions if you are confident of your skills.

A tutor will check the Discussion Forum regularly, and answer some questions, but be aware that for some questions you will just need to use your judgment and document your thinking. For example, a question like, "How much data should I use for the experiments?", will not be answered; you must try out different data and see what makes sense.

Submission

Your C code files (including your `Makefile` and any other files needed to run your code) should be submitted through the LMS under **Assignment 1: Code** in the **Assignments** tab.
Your programs *must* compile and run correctly on JupyterHub. You may have developed your program in another environment, but it still *must* run on JupyterHub at submission time. For this reason, and because there are often small, but significant, differences between compilers, it is suggested that if you are working in a different environment, you upload and test your code on JupyterHub at reasonably frequent intervals.

A common reason for programs not to compile is that a file has been inadvertently omitted from the submission. Please check your submission, and resubmit all files if necessary.

Assessment

There are a total of 15 marks given for this assignment.

Your C program will be marked on the basis of accuracy, readability, and good C programming structure, safety and style, including documentation (2 marks). Safety refers to checking whether opening a file returns something, whether mallocs do their job, etc. The documentation should explain all major design decisions, and should be formatted so that it does not interfere with reading the code. As much as possible, try to make your code self-documenting, by choosing descriptive variable names. The remainder of the marks will be based on the correct functioning of your submission.

Plagiarism

This is an individual assignment. The work must be your own.

While you may discuss your program development, coding problems and experimentation with your classmates, you must not share files, as this is considered plagiarism.

If you refer to published work in the discussion of your experiments, be sure to include a citation to the publication or the web link.

"Borrowing" of someone else's code without acknowledgment is plagiarism. Plagiarism is considered a serious offense at the University of Melbourne. You should read the University code on Academic integrity and details on plagiarism. Make sure you are not plagiarizing, intentionally or unintentionally.

You are also advised that there will be a C programming component (on paper, not on a computer) in the final examination. Students who do not program their own assignments will be at a disadvantage for this part of the examination.

Late policy

The late penalty is 10% of the available marks for that project for each day (or part thereof) overdue. Requests for extensions on medical grounds will need to be supported by a medical certificate. Any request received less than 48 hours before the assessment date (or after the date!) will generally not be accepted except in the most extreme circumstances. In general, extensions will not be granted if the interruption covers less than 10% of the project duration. Remember that departmental servers are often heavily loaded near project deadlines, and unexpected outages can occur; these will not be considered as grounds for an extension.

Students who experience difficulties due to personal circumstances are encouraged to make use of the appropriate University student support services, and to contact the lecturer, at the earliest opportunity.

Finally, **we are here to help!** There is information about getting help in this subject on the LMS. Frequently asked questions about the project will be answered in Piazza.