THE UNIVERSITY OF SYDNEY

COMP2129 Assignment 4

Bookworm

Assignment Due: 11:59pm Friday, 9 June, 2017

This assessment is worth 10% of the course

Task Description

You are tasked with writing a program that will perform queries on a graph of books. The books themselves are represented as an array of book_node_t and will contain categorised edges linking to other books. These queries will require you to develop a parallelism strategy depending on the query requirements.

Some queries may require simple partitioning of data for each thread to operate on, while others may require strategy of dynamically partitioning data, checking the distribution of the data and synchronisation in order to efficiently process and eliminate redundant operations.

You can ask questions on Ed using the assignments category. Please ensure that your work is your own and you do not share any code or solutions with other students.

Working On Your Assignment

You can work on this assignment using your own computers or the lab machines. Students can also take advantage of the online code editor on Ed. You will need to navigate to the assessments or workspaces page where you will be able to run, edit and submit your code from within your browser. However we recommend you write this code using your own computer or the lab machines.

It is important that you continually back up your assignment files onto your own machine, flash drives, external hard drives and cloud storage providers. You are encouraged to submit your assignment while you are in the process of completing it to receive feedback and to check for correctness of your solution.

Academic Declaration

By submitting this assignment you declare the following:

I declare that I have read and understood the University of Sydney Academic Dishonesty and Plagiarism in Coursework Policy, and except where specifically acknowledged, the work contained in this assignment or project is my own work, and has not been copied from other sources or been previously submitted for award or assessment.

I understand that failure to comply with the the Academic Dishonesty and Plagiarism in Coursework Policy, can lead to severe penalties as outlined under Chapter 8 of the University of Sydney By-Law 1999 (as amended). These penalties may be imposed in cases where any significant portion of my submitted work has been copied without proper acknowledgement from other sources, including published works, the internet, existing programs, the work of other students, or work previously submitted for other awards or assessments.

I realise that I may be asked to identify those portions of the work contributed by me and required to demonstrate my knowledge of the relevant material by answering oral questions or by undertaking supplementary work, either written or in the laboratory, in order to arrive at the final assessment mark.

I acknowledge that the School of Information Technologies, in assessing this assignment, may reproduce it entirely, may provide a copy to another member of faculty, and or communicate a copy of this assignment to a plagiarism checking service or in house computer program, and that a copy of the assignment may be maintained by the service or the School of IT for the purpose of future plagiarism checking.

Structure Details

Your program will be compiled without any optimisations -O0. You are allowed to use compiler intrinsics and other features for students who would like to take advantage of such features available.

We will provide the graph array but you will need to implement the code required to utilise this data. We will also specify the number of threads that your query is limited to spawn.

```
typedef struct book_t {
        size_t id;
        size_t author_id;
        size_t publisher_id;
        size_t* b_author_edges;
        size_t* b_citation_edges;
        size_t* b_publisher_edges;
        size_t n_author_edges;
        size_t n_citation_edges;
        size_t n_publisher_edges;
} book_t;
```

The size_t* elements are indexes referring to other books within the same structure, creating a simple link between books.

For all queries you will need to return a result_t structure: This structure stores all elements retrieved from the query and the number of elements. It will be used to determine if your is correct.

```
typedef struct result_t {
        book_t** elements;
        size_t n_elements;
} result_t;
```

Each book node will link to other book nodes through one of three of the types of edges (authors, citations, publishers).

The graph structure (library) itself may contain **multiple books** of the **same id** and you cannot assume that a book is unique from its id alone. However, if a book has the same id and exists multiples times in the graph you can **assume** that it has the same **citations references** and **author** links.

Queries

You can think of these similar to SQL select queries on an unindexed table. Some queries are easy to partition while others may require you to employ a more specific strategy to equally distribute work between threads or to synchronise threads between others.

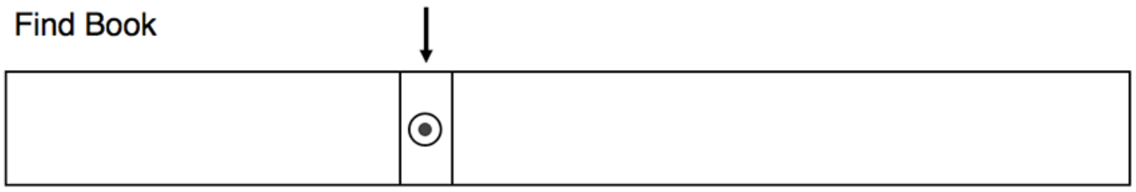
The diagrams shown will contain multiple bars, this should not be confused with different arrays, these are operations that are performed on the **same array of data**.

Find a book

Searching through the graph to find a specified book.

```
result_t* find_book(book_t* nodes, size_t count, size_t book_id);
```

Given an array of nodes and the count, your query will need to find a book by the id given in the function. Return only **one** instance of the book.

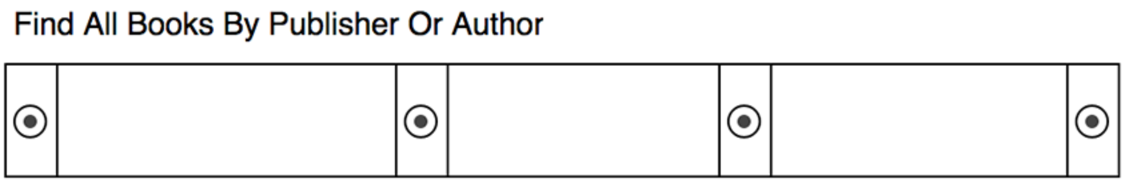


Find all books by author

Searching through all the books and returning the books that have the same author.

```
result_t* find_books_by_author(book_t* nodes, size_t count, size_t author_id);
```

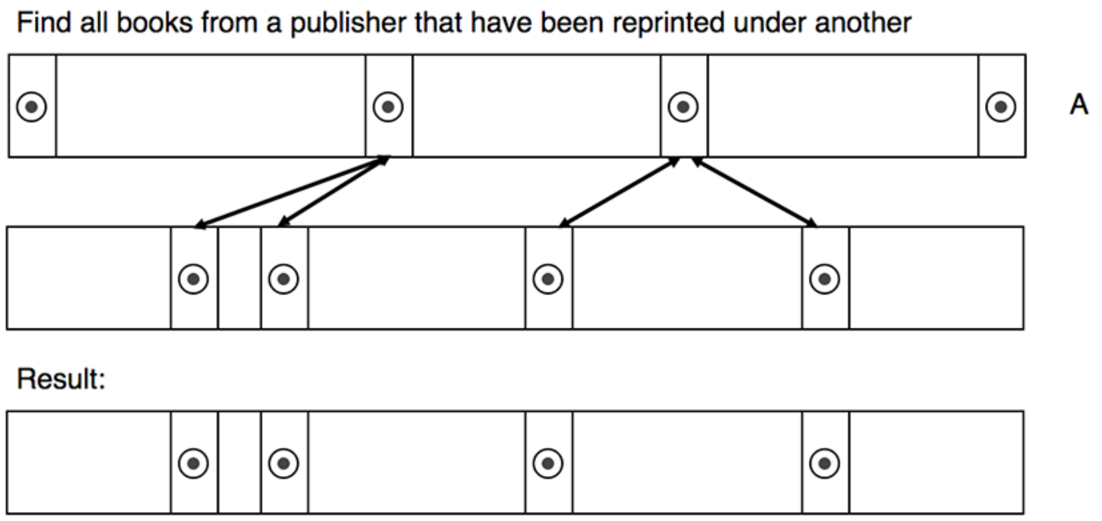
Given an array of nodes and the count, your query will need to return all books that are associated to an author.



Find all books that have been reprinted under another publisher

```
result_t* find_books_reprinted(book_t* nodes, size_t count, size_t publisher_id);
```

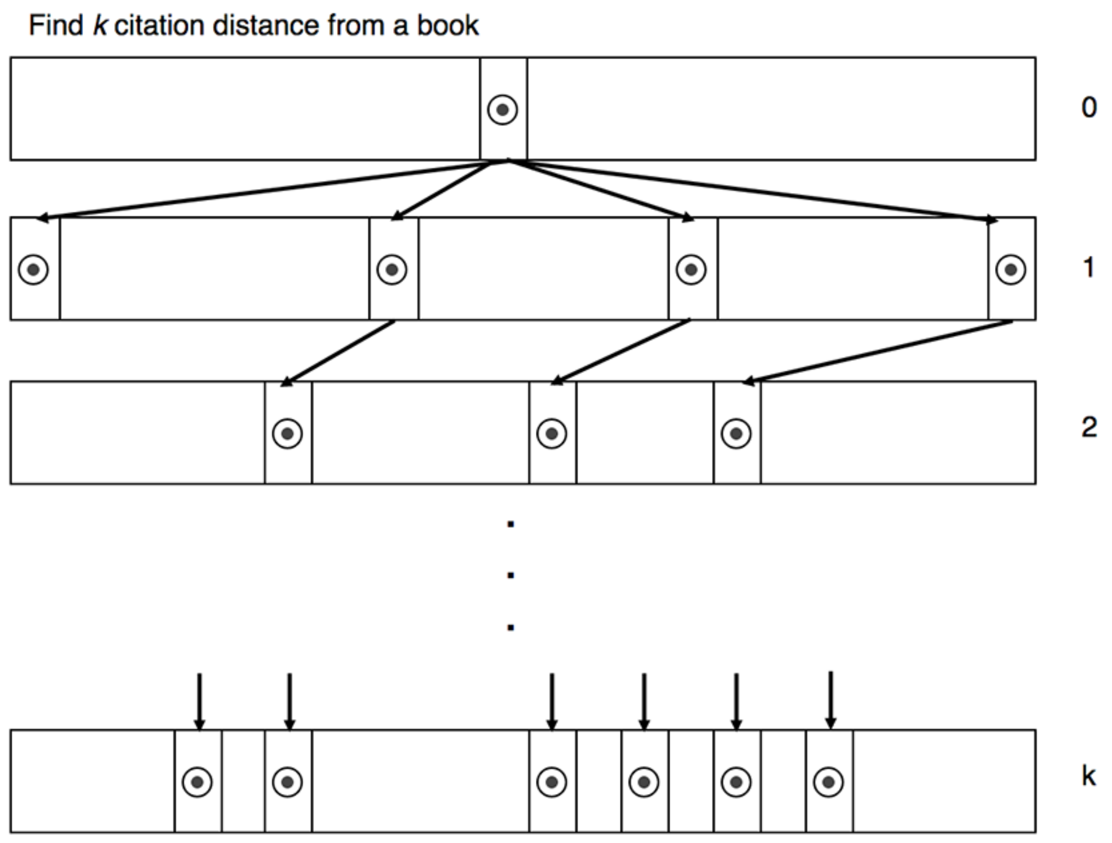
Given an array of nodes and the count, your query will need to return all **reprinted** copies of a book. You may assume that the publisher_id that is given is the source.


Find all books from a publisher that have been reprinted under another
A
Result:

Find all books of k degree of separation of a book

Your query will need to find a specific book (which is included in the result and is also considered level 0) and find all other books related only via the citation edges. These books must be within k distance of the found book. Your query must only use citation edges and find k degree of separation from book 0.

```
result_t* find_books_k_distance(book_t* nodes, size_t count, size_t book_id,
  uint16_t k);
```



Find the shortest distance between two books

```
result_t* find_shortest_distance(book_t* nodes, size_t count,
  size_t b1_id, size_t b2_id);
```

Your query will need to determine the shortest distance between two books. You will need to factor in all edges from a book node to other book nodes to find the shortest distance. Your query will need return the book nodes it visits to reach **book b** from **book a** inclusive of **b** and **a**. In the event that your algorithm finds two or more shortest paths between **a** and **b** of the same length, your algorithm just needs to return one of them.

What if the books specified in the query (book1_id and book2_id) are the same?

If two books share the same id, you should return both books.

What if book1_id occurs in the graph more than once?

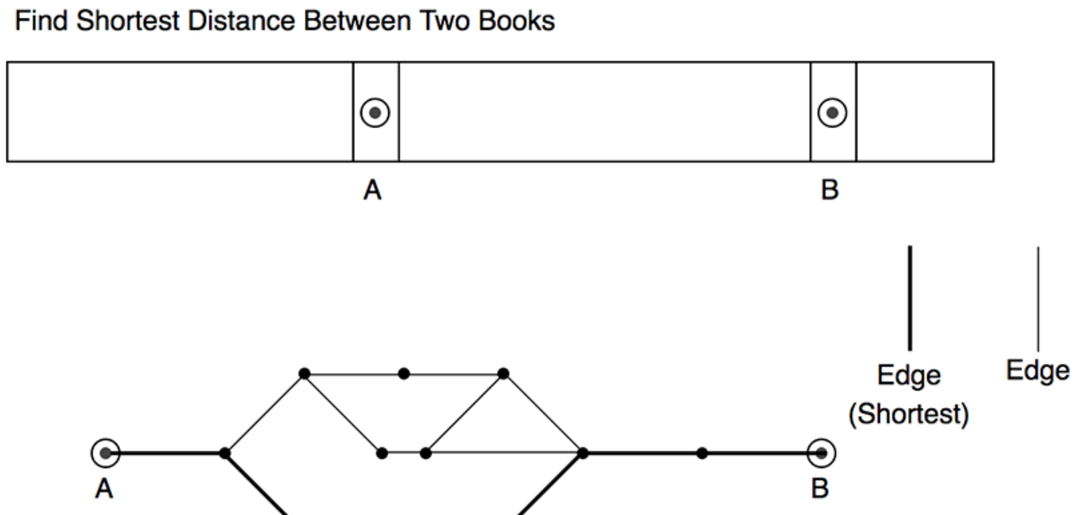
You only need to return one of the books, this also applies with book2_id. Next part covers reprints.

What about reprints?

Potentially there is a case where both books can contain a reprint, if this is the case, there is a chance that reprint's publisher edges could be shorter. You will need to factor this into your query, We'd recommend focusing on just getting a basic implementation and assume that there is no reprints as dealing with this case is trivial once everything else is working.

What if b1_id and b2_id are the same and only one instance of a book exists in the graph?

Your query should return only one element that was found in the graph.



Algorithms

You may implement each of these algorithms depending on what query you want it used for.

Breadth First Search

Breadth first search will check its neighbours before moving onto its neighbour's neighbours and so on.

```
BFS(G, v)
    Queue q;
    for all vertices u in G
        u.seen = false
    q.enqueue(v)
    while(!q.empty())
        u = q.dequeue()
        if u.seen is false
            u.seen = true
            for all neighbours w of u
                q.enqueue(w)
```

Depth First Search

Depth first search starts at some vertex in the graph and traverses the graph as far as it can before backtracking and traversing neighbours of that node.

```
DFS(G, v)
    Stack s;
    for all vertices u in G
        u.seen = false
    s.push(v)
    while(!s.empty())
        u = s.pop()
        if u.seen is false
            u.seen = true
            for all neighbours w of u
                s.push(w)
```

You may apply a Breadth First Search or Depth First Search depending on what you are familiar with or what you think is appropriate. You are also free to implement something different if it fits your need.

Testing and Submission Details

You will be provided with simple benchmarking code for this assessment called worm_bench and its source. It will time your query as well as load any expected results you have stored. Your program must produce no errors on the lab machines and Ed and will be compiled with the command:

```
clang -O0 -std=gnu11 -march=native -lm -lpthread
```

You are to submit your assessment using Ed which will check for correctness of your queries. In the event that your program does not produce the correct output your program will not checked for performance.

Marking

3 Marks assignment for the correctness of your program. This requires implementing all queries specified in the assignment and ensuring that they will return the correct result.

5 Marks are assigned based on the performance of your code related to the benchmark. This is tested on a separate machine. Submissions that are faster or equal to the benchmark set, you will receive full marks. Submissions faster than a basic implementation will receive a minimum of 2.5 marks.

2 Marks are assigned for a manual marking component in your tutorial. This will be based on your submission prior to the **5th of June 2017 10:00am AEST**.

This manual marking component will assess your current progress of your assessment, styling (indentation and function layout) and also explaining your code to your tutor.

Warning: Any attempts to deceive or disrupt the marking system will result in an immediate zero for the entire assignment. Negative marks can be assigned if you do not properly follow the assignment specification, or your code is unnecessarily or deliberately obfuscated.