COMP2017 / COMP9017 Assignment 2

Due: June 08th, 2pm AEST (Week 13 Friday)

This assignment is worth 10% of your final assessment

Task Description

The given data has been collected from a social media platform supergraph2048. You are tasked with writing program using the C programming language that will handle different queries and compute the correct result.

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.

Program structure

The social media platform is composed of two types of entities:

- Users
- Posts

Each user in the social network have a list of followers, other users they are following and posts. These collections are represented as an array of indices that relate to a `user*` or `post*` array.

```c
struct user {
  uint64_t user_id; //User id, is unique
  size_t* followers_idxs; //User indices within user* users graph
  size_t n_followers; //Number of followers
  size_t* following_idxs; //User indices within user* users graph
  size_t n_following; //Number of users the user is following
  size_t* posts_idxs; //Post indices within post* posts graph
  size_t n_posts; // Number of posts a user has made.
};
typedef struct user user;
```

All `user_id` values within the users array will be unique. The lists: followers, following and posts are arranged by ascending order. In order of insertion.

```c
struct post {
  uint64_t pst_id; //Post id, is unique
  size_t* reposted_idxs; //Reposted indices within the graph
  size_t n_reposted; // Number of reposted posts.
};

typedef struct post post;
```

All `pst_id` values within the posts array will be unique. The posts array will not contain any cycles as for any reposts to exist depends on the existance of an original post.

For all queries you will need to return a heap allocated `result` structure: This structure stores all elements retrieved from the query as a heap allocated contiguous block of pointers and the number of elements. It will be used to determine if your result is correct.

```c
struct result {
  void** elements;
  size_t n_elements;
};

typedef struct result result;
```

The entire social network structure is an array of users and an array of posts. Both elements contain links to each other as well as to themselves.

Setup

Your program will have opportunity to set up threads or processes to be utilised within your program. We will take into account the setup time required as a separate benchmark. Your program will need to be prepared to handle **consecutive queries** during its execution.

Given the following struct declaration:

```
struct query_helper {
        int dummy_field;
        //YOUR OWN FIELDS
};

typedef struct query_helper query_helper;
```

Your can declare any pointers or data that you may use in your query helper struct for your queries. If you do not wish to use this structure and instead create threads/processes during the query itself, you will need to leave the `dummy_field` in there.

`query_helper* engine_setup(size_t n_processors)`

You are given an empty definition of this function with your scaffold. You may modify this function for your own purposes. Once you have returned a `query_helper` object, it will be used in the rest of the queries.

After the program has finished executing queries, your helper will need to be cleaned up with the following function. You will need to dispose of any memory that you have allocated to this helper.

`void engine_cleanup(query_helper* helper)`

Example:

```
query_helper* helper = engine_setup(N_PROCESSORS);

find_all_reposts(posts, n_posts, post_id_1, helper);
find_all_reposts(posts, n_posts, post_id_2, helper);

engine_cleanup(helper)
```

If your `engine_setup` function takes too long. Your program will be stopped by the automarking system.

Find all reposts (Query)

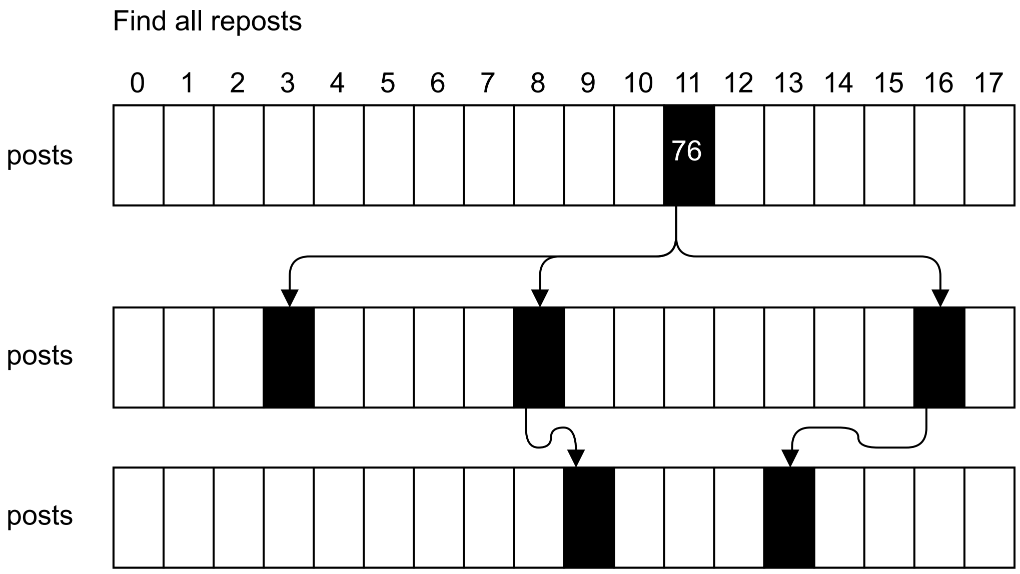
Given a post id, find all reposts of the given post (this can also be reposts of a repost and so on). The query should return an array of all reposts and originating post, within the `result` structure and `n_elements` is set appropriately. If no reposts have been made, then the query should return the post which the post id corresponds to.

If the post id does not exist in the set, your result structure elements should be NULL and `n_elements` is 0.

```
result* find_all_reposts(post* posts, size_t count,
                    uint64_t pst_id, query_helper* helper)
```

For example:

```
find_all_reposts(posts, 18, 76, helper)
```

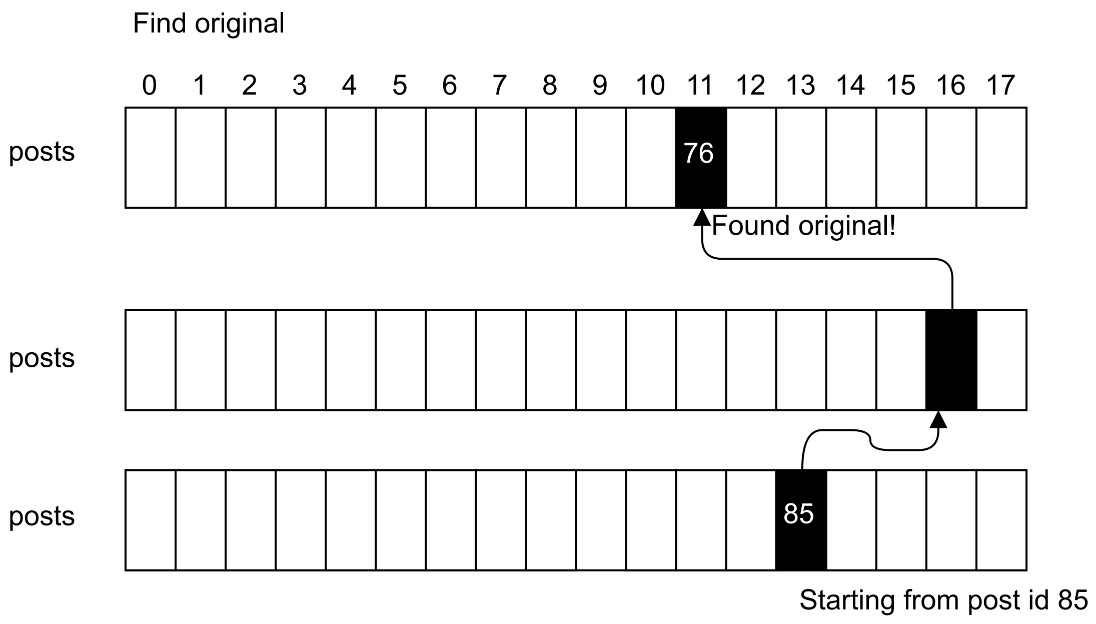


Find original post (Query)

Given a post id, your query will need to initially determine if the post is an original or a repost. If it is an original, your query will return the post. Otherwise, your query will search until it finds the original post. The query should only return one element which is the original post.

If the post id does not exist in the set, your result structure elements should be NULL and `n_elements` is 0.

```
result* find_original(post* posts, size_t count,
        uint64_t pst_id, query_helper* helper)
```



Find the shortest path between two users (Query)

Given two users within the social network. Your query will need to find the shortest path between two users based on who they are following. We define outgoing edges as all users referred in the `following_idxs` property in the `user` struct. Incoming edges are defined as all users referred in the `followers_idxs` property in the `user` struct.

The shortest path is defined as the minimum number of edges to between a source and destination.

Your query will need to consider that there may exist a shortest path between either:

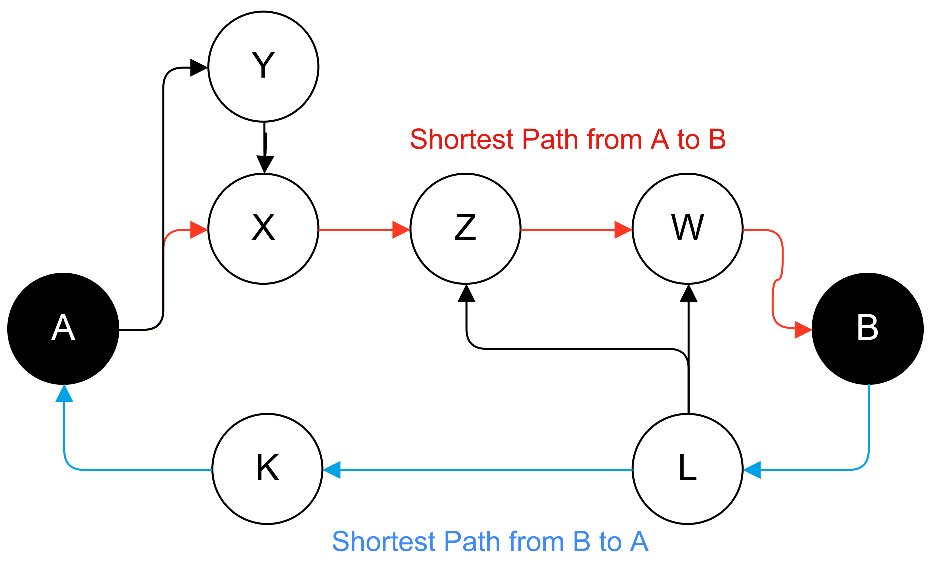
- **userA** and **userB**, based on outgoing edges starting from **userA**
- **userB** and **userA**, based on outgoing edges starting from **userB**

In the event that your algorithm finds two or more shortest paths, Your query requires returning just one of them. Your query must return a result including all users (including of **userA** and **userB**) that compose the shortest path.

Your query must return `NULL` if the users specified in the query (userA and userB) are the same, or either userA or userB does not exist.

```
result* shortest_user_link(user* users, size_t count,
        uint64_t userA, uint64_t userB, query_helper* helper)
```

Each outgoing edge corresponds to an element in "following" for a user



Shortest path is between B and A

Find all the robot accounts (Query)

A common problem in social networks is the proliferation of robots. These robots unfortunately can create a large amount of sway in the political climate. You are tasked with utilising heuristic data to classify and return all robots.

```
struct criteria {
        float oc_threshold;
        float acc_rep_threshold;
        float bot_net_threshold;
};

typedef struct criteria criteria;
```

The criteria struct defines three floating point numbers which will be within the range of $[0, 1]$. If any of the fields of the struct contains a value outside of an acceptable range, your query must return an empty result struct (`n_elements` is 0, `elements` is `NULL`).

```
result* find_bots(user* users, post* posts, size_t user_count,
        size_t post_count, criteria* crit, query_helper* helper)
```

A robot account will:

- **Reposts more than posts**, Bots typically do not generate their own content and instead repost other posts. Given the threshold specified in the criteria, your query needs to determine if a user is a bot.

$$\frac{reposts}{n\_posts} > oc\_threshold$$

- **Account reputation**, You will need to acknowledge the user reputation, caluclating it based on:

$$\frac{n\_followers}{(n\_followers + n\_following)} < acc\_rep\_threshold$$

  If a user is considered to be a bot that **reposts more than posts** or has an odd **account reputation**, you are to include this user in your result.

  After initially discovering bots within the network using the previous criteria, you can use this data to find **discrete bots**.

- **Finding discrete bots within a network**, Political bots like to give certain posts legitimacy through popularity. Unfortunately this may mean that certain bots may not adhere to any of the previous characteristics. This is because they are designed to create content other bots will propergate through the network.

  We can classify a discrete bot by checking

$$\frac{bots\_following}{n\_followers} > bot\_net\_threshold$$

Algorithms

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)
```

You may apply a Breadth First Search or any other algorithms based on what you think is appropriate.

Error handling

For all four queries, you will need to ensure that your query returns **heap allocated memory**. If the data set provided is `NULL` or the number of elements (`count`) is 0, your query must return an empty result set (`n_elements` is 0, `elements` is `NULL`).

Submission Details

You will be provided with benchmarking code for this assessment called `supergraph2048_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 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 be checked for performance.

- **4 Marks** for the correctness of your program. This requires implementing all queries specified in the assignment and ensuring that they will return the correct result. If it does not compile, it will receive zero.
- **4 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, will receive full marks. Submissions faster than a basic implementation will receive a minimum of **2 marks**.
- **2 marks** are assigned for a manual marking component in your tutorial. This will be based on your submission prior to the **4th of June 2018 8:00am 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 follow the assignment specification or if your code is unnecessarily or deliberately obfuscated.

Academic declaration

By submitting this assignment you declare the following:

I declare that I have read and understood the University of Sydney Student Plagiarism: Coursework Policy and Procedure, and except where specifically acknowledged, the work contained in this assignment/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 Student Plagiarism: Coursework Policy and Procedure 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 acknowledgment 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.