`OA` : Offset Array.

`NA` : Neighbor Array.

In our implementation, the graph is in CSR format. Which means that, if you want to know the neighbor of `node i` , get `OA[i]` and `OA[i+1]` , the index range of `NA` that stores neighbors.

In the `graph.size` file, it has 8 bytes, which are the number of nodes and number of edges. (`|nodes|` , `|edges|`)

In the `graph` file, the first `|nodes|+1` *4 Bytes are values of `OA` . The following `|edges|` *4 Bytes are values of `NA` . The last `|edges|` *4 Bytes are values of edge weight for corresponding edge.

Dataset Download

You can get dataset, and sample output of Assignment 1 from the following link:

CSC3150_23-24Term2_A1_data - OneDrive

Your dataset should be placed at the same directory of code

```
main@ubuntu:~/Desktop$ ls
a.out  grader  g1  g1.size  g2  g2.size  graph.cpp  graph.h  sample_output
```

Important

In this assignment, all datasets to read and output files you need to write are in binary format. Please take a look at `std::ios::binary` .

sample_output is used for self-checking if the answers to task1 are correct.

Task1

In task1, you are provided with data sets `graph1` , `graph2` correspondings to `Task1_1` and `Task1_2` . In such graphs, the nodes can be represented as cities, edges between cities can be represented as roads. We provide a program traversing nodes through BFS algorithm.

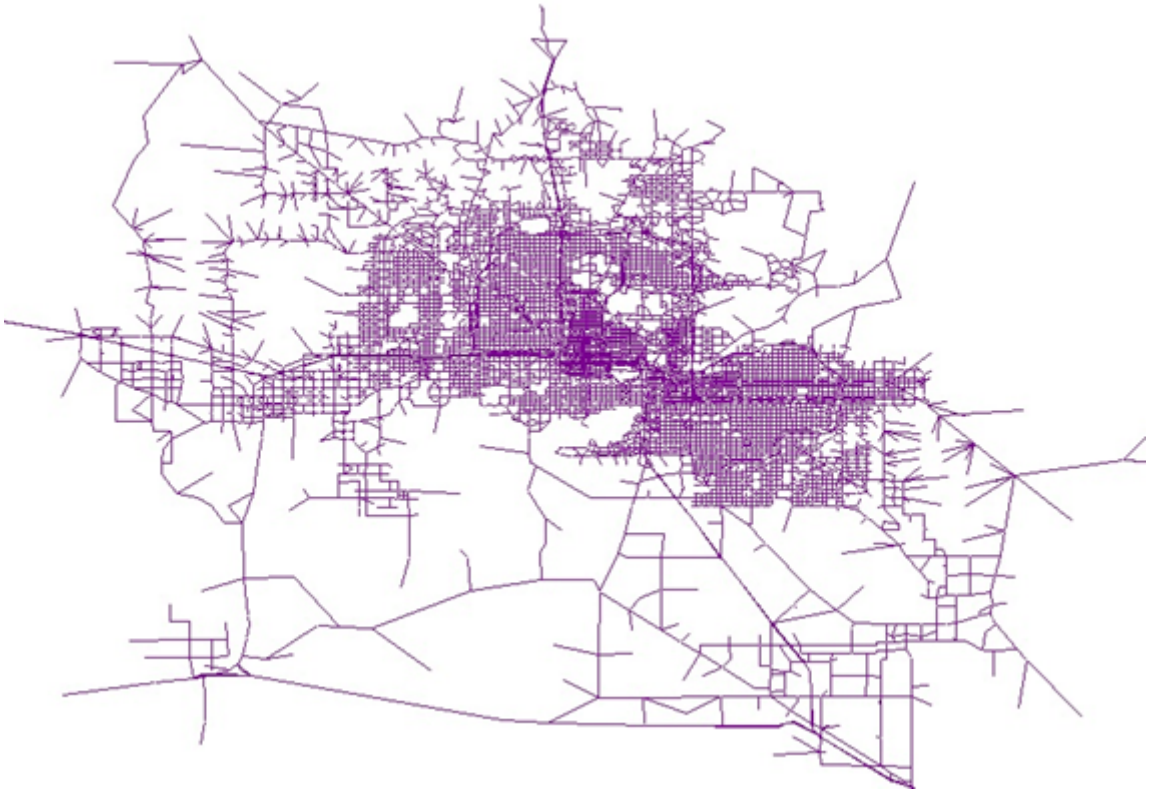
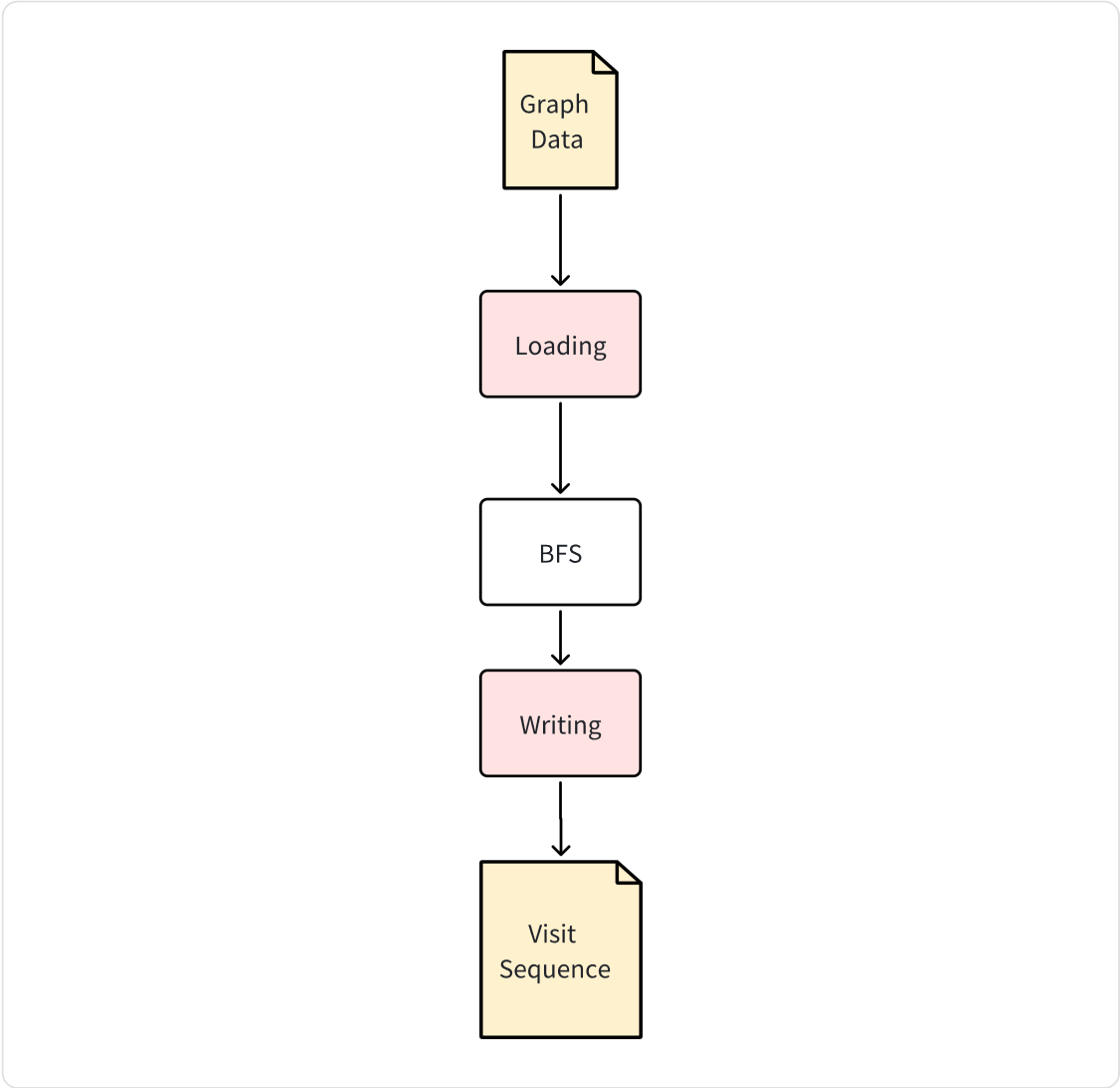


Figure: Transportation Graph

- Your first task is to complete the remaining part of program so that it can correctly load data into memory and do BFS traverse execution.
- After BFS traversal is done, the program needs to store the sequence of visits to the result file.
- In the following progress charts, blocks in **yellow** means it's a file, in **red** means it's a **TODO** part for you.
- In this task, you need to complete `Task1_1(), Task1_2()`. In `Task1_2()` we only have 2GB memory but `g2` has 1.8GB data. Traditional read from disk to memory does not work.



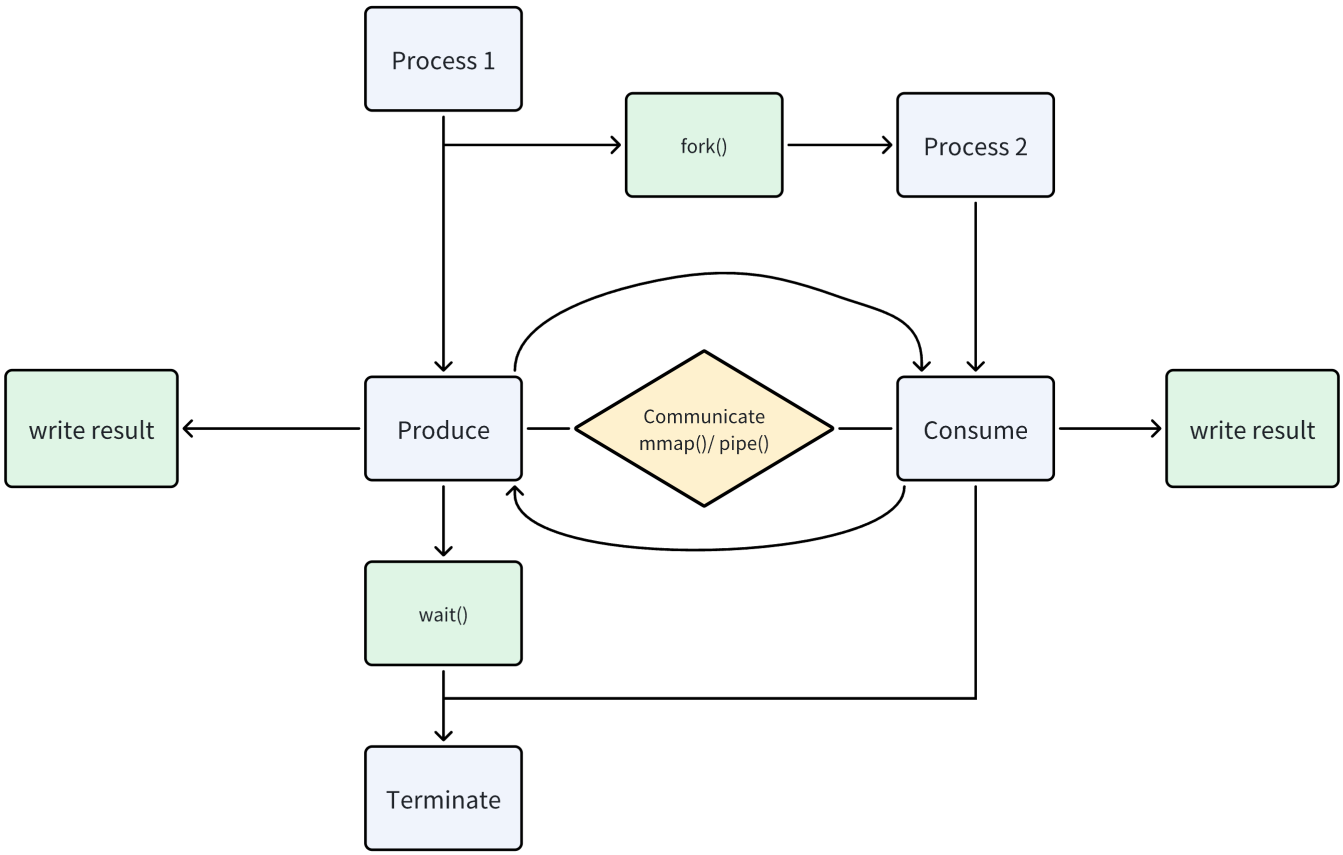
Hint

- When loading the graph, please note that `OA` has length `|nodes|+1` instead of `|nodes|`. This is because we miantain one more integer for some boundary problems. What you read from the `.size` file is `|nodes|` and `|edges|`.

Task2

In task 2, the program goes into a simple simulation. In the daytime, a node gains weights from its edges with corresponding edge weights, implemented by `produce()`. In the night, a node's weight will self-increased by 1, implemented by `consume()`.

The global view of process is shown in the figure below. We highlight the components you needs to do in green and yellow blocks:



You are required to use `fork()` to handle two processes. For the parent process, we will do calculation of `produce()`. For the child process, we will do `consume()`. Both of these functions are provided, you do not need to do any modification to it.

```
// Executed in parent process
void produce(int *weights, int len, Graph &g) {
    for (int i=0; i<len; i++) {
        int l,r;
        g.get_edge_index(i,l,r);
        for (int j=l;j<r;j++) {
            weights[i+1]+=g.edge_weights[j];
        }
    }
}


// Executed in child process
void consume(int *weights, int len, Graph &g) {
    for (int i=0; i<len; i++) {
        weights[i+1] += 1;
    }
}
```

TODO

- Your task is to complete the following functions with correct system call and control logic so that program runs correctly as shown in the above figrue.
- In implementation of communication, it actually shares the data stored in `weights`
  - Format of array `weights` : the first integer should be the value of current iteration (start from 0). The following `|V|` integers are corresponding weights of vertex
- For each iteration generated by child process, child process should write result to file

  e.g. For 4th and 5th iterations, write sequence of nodes weight to `/home/csc3150/A1/Task2_parent.output_2` and `/home/csc3150/A1/Task2_child.output_2`.
- Program terminates at 10th iteration

```
// Task2: Process Inter-Communicationvoid Task2() {
    Graph g;
    int fd;
    fd = g.map_global_graph("g2");
    std::string ipc_path("ipc_file");
    // Creating inter process communication file if there is not.
    int ipc_fd = open(ipc_path.c_str(),O_RDWR| O_CREAT,0777);
    lseek (ipc_fd, (g.v_cnt+1)*sizeof(int)-1, SEEK_SET);
    write (ipc_fd, "", 1);
    close(ipc_fd);
    std::string output_parent_path("Task2_parent.output");
    std::string output_child_path("Task2_child.output");
    // process control
}

// Task2: Process Inter-Communication with control
void parent_process(const std::string &path) {
    int pid = getpid();
    printf("parent proc %d is producing\n",pid);

    // produce()
    return;
}

// Task2: Process Inter-Communication with control
void child_process(const std::string &path) {
    int pid = getpid();
```

```
        printf("child proc %d is consuming\n",pid);

        // consume()
        return;
    }
```

Hint

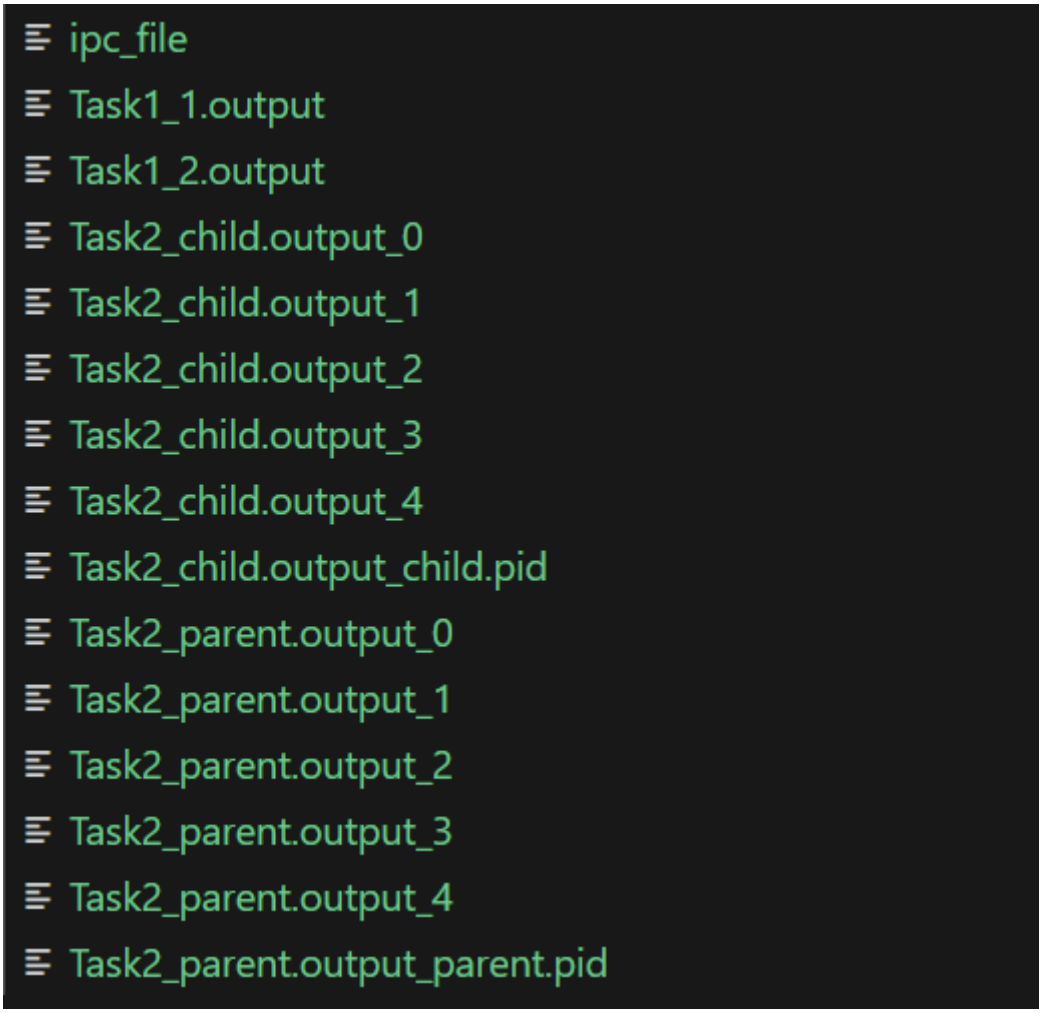
- The status of vertices need to be communicated inter-process at each iteration because both processes need to process/ fetch data.
- You might need to define `index 0` of mapped file in mmap() as control `int`
    - Because of Data Dependency, execution order matters
    - By reading control byte, each process will know whether they should wait, process, or terminate.
- You can use mmap() to share reading the data graph if both processes need information of graph.
    - Try to figure out how memory is consumed when multi-process using mmap map to one file.

Output files

You are expected to generate the following files when you complete all the tasks

All these files are in the same directory as ***graph.h, graph.c***

(wrong file name will not be correctly graded):



Report 10'

You shall strictly follow **the provided latex template** for the report, where we have emphasized important parts and respective grading details. **Reports based on other templates will not be graded**.

Report Template

- You can find latex template in the following link
  - https://www.overleaf.com/read/ybgnjwnyvjpx#dd17ed
- We also provide template at BB.

LaTex Editor

For your convenience, you might use Overleaf, an online LaTex Editor.

1. Create a new blank project.
2. Click the following highlight bottom and upload the template we provide.
3. Click **Recompile** and you will see your report in PDF format.