2021 SEMESTER 1
COMP20007 DESIGN OF ALGORITHMS
SCHOOL OF COMPUTING AND INFORMATION SYSTEMS
THE UNIVERSITY OF MELBOURNE

Assignment 2

Due: 5:00 PM AEST Thursday 27th May

Introduction

This assignment consists of 3 problems: 1 programming problem and 2 written problems. You will submit your solutions for the C programming component via `dimefox submit` and the written component via the LMS.

This assignment has a total of 20 marks and will contribute 20% to your final grade for this subject.

Programming Problems

Problem 1

1 + 2 + 2 + 2 + 1 + 2 + 2 = 12 Marks

Australia faces one of its biggest disasters: a total power outage that lasted for a few minutes. You are the CTO for the National Super Network (NSN). NSN is responsible for Australia's Internet backbone and connects all key Internet servers in Australia. After the blackout you decide that it is best to ascertain the consequences of the power outage.

NSN has built its high-speed network so that

- all server connections are bi-directional;
- any two servers are connected if they share an edge or each of them has a path to a shared server;
- the network can have subnetworks.

If everything is working well, all servers in a subnetwork are connected. Fortunately, each server can be directly contacted through a slow telephone connection. You need to develop an algorithm that determines the number of connected components in the network.

Task 1

1 Mark

Develop the pseudocode for an efficient algorithm that computes the number of connected components in a graph $G = (V, E)$ given vertices V and edges E. Discuss the complexity of your algorithm in detail.

Task 2

2 Marks

Write a C program that takes the name of one file as a command line argument containing the specification for the network, reads in a text file from standard input and prints out the total number of connected subnetworks before the outage for this network. Suppose that the network has n servers and each server has a unique server ID (SID) which is an integer between 0 and $n - 1$ inclusively.

The file specified as the command line argument for this and the subsequent programming tasks has two parts:

- The first line gives the number of servers, n, and the number of network connections in the graph, m, separated by a space.
- The following lines specify the m connections between servers, one per line, given as the two SIDs separated by a space. Note that each edge is listed only once.

The text file received on standard input is also given in two parts:

- The first line specifies the number of servers which are affected by the outage.
- The following line contains a list of all the SIDs affected by the outage, each separated by a space.

An example input text file (`network-1.txt`) is given below. The network configuration before the power outage is demonstrated in the following figure.

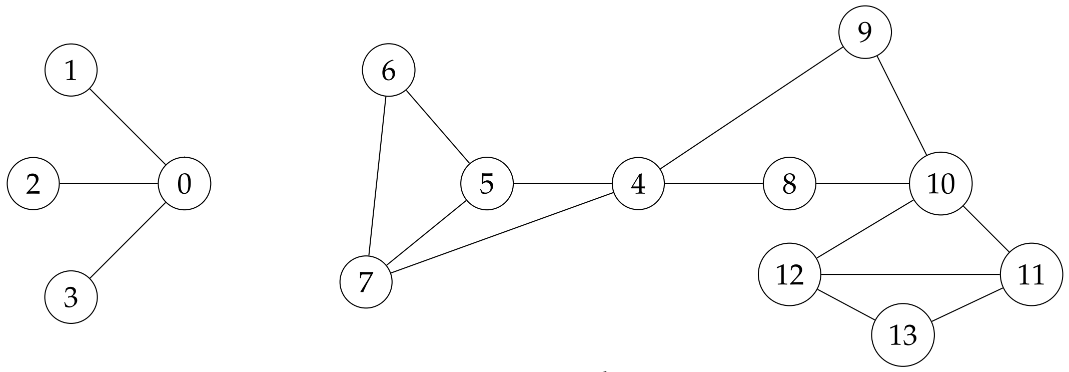

Figure 1: network-1.txt

All four provided network test cases are shown at the end of the document.

You can assume that the text files used to test your program will be always sensible and correct. You do not need to perform any data validation. You can assume that a connection between any pair of servers appears only once in the list. You can also assume the source and destination server of a connection is always different.

```
    14 17
    0 1
    0 2
    0 3
    4 8
    4 9
    4 5
    4 7
    5 6
    5 7
    6 7
    10 8
    10 12
    10 11
    10 9
    11 12
    11 13
    12 13
```

An example of the text file sent in through standard input (`outage-1.txt`) is:

```
    2
    1 4
```

The required output for the above pair of input files is:

```
Before the outage, the number of connected subnetworks is: 2
```

To get marks for this task, the output of your program has to be exactly as shown above, with at least one whitespace before the number with no other output printed. You must have a program compiled for this task using `make task2` which, for this example, must run with `./task2 tests/network-1.txt < tests/outage-1.txt`.

In this task, as well as in the subsequent programming tasks, when there are multiple equally possible servers always choose the one with the smallest ID.

Task 3

2 Marks

Amend your C code to now also produce `task3`, a program which computes the largest subnetwork *before* the power outage. The largest subnetwork is the connected subnetwork that has the largest number of connected servers. Print out the total number of servers and the server IDs of all servers in that subnetwork in *increasing order*. If the largest subnetwork is not unique, print out the one containing the smallest SID.

The required output when run with the input files `./task3 tests/network-1.txt < tests/outage-1.txt` is:

```
Before the outage, the number of servers in the largest subnetwork is: 10

The servers in the largest subnetwork are: 4 5 6 7 8 9 10 11 12 13
```

Note: As part of developing task3 and all following tasks, your code should still allow the compilation of a working solution to Task 2.

Task 4

2 Marks

Amend your C code to now also produce `task4`, a program which computes the largest diameter found in any of the subnetworks *after* the power outage. The diameter of a graph (network) is defined as the longest shortest path between any pair of two vertices. Print out the length of the diameter and the server IDs of all servers on that path in the order they appear along the path. The length of a path is defined as the number of edges in that path. If that path is not unique, print out the one that starts from the server with the smallest ID, and if they have the same starting server, print the one with the smallest ID of the last server.

The required output when run with the input files `./task4 tests/network-1.txt < tests/outage-1.txt` is:

```
After the outage, the largest diameter in any of the subnetworks is: 3
```

```
The path is: 8 10 11 13
```

Note: As before, your solution to task 4 should not affect the correctness of your solutions to earlier tasks.

Task 5

1 Marks

To ensure that a similar disaster cannot happen again, you decide that we have to identify all servers that will increase the number of subnetworks if they fail. We call these servers *critical servers*. You realize that these servers need to be secured against power failure to prevent a similar outage.

Develop the pseudocode for an algorithm to compute all critical servers (or rather critical vertices) in a network (graph). A vertex is called *critical* if the removal of the vertex and its associated edges increases the number of connected components of the graph.

In this task, you are required to use a brute force approach. Namely, you will check for every vertex if it is a critical point by removing it and counting the number of connected components in the graph. Discuss the computational complexity of your algorithm in detail.

Task 6

2 Marks

You realize that your own algorithm for computing all critical vertices is too slow. You ask your friend Jan Art who has just finished their computer science degree. Jan Art suggests that you could use DFS traversal to implement an algorithm in $O(V + E)$ time. The key concept to use is the *back edge*. Fortunately, you remember from your lectures how a back edge is defined. The key insight is that the discovery of a back edge (when constructing a DFS tree) indicates an alternative path for a vertex (and its children) in case the parent of that vertex is removed.

If for a vertex v with a parent vertex u in a DFS tree of a connected component, we cannot find any back edge for any of the vertices in its subtree, then the removal of the parent vertex u would imply that there is no path from u or from any ancestor of u to v or to any descendants of v. That is, v and its subtree would become disconnected and is no longer part of the connected component. Given a DFS tree and vertices u and v in that tree, we say that u is an *ancestor* of v and v is a *descendant* of u, if there is a path from u to v.

Suppose that u is an ancestor of v in the DFS tree. If there is a back edge from v to u, we say that u is a *reachable ancestor* of v. For each v, we examine the set of all reachable ancestors of all vertices in the subtree rooted at v, and identify the one with the smallest push order as the *highest reachable ancestor* (HRA) of v. If there is no reachable ancestor in the whole subtree, we simply set HRA to v.

The idea of the algorithm is as follows: we apply DFS and record for every vertex v its push order and its HRA. If a vertex u has a child v such that the push order of v's HRA $\geq$ than the push order of u, then u is a critical vertex.

This only applies if a vertex u is not the root or a leaf of the DFS tree. If it is the root, it suffices to check if u has more than more one child. If it does, then u is a critical vertex. On the other hand, if u is a leaf, it is not a critical vertex.

Write the pseudocode for Jan Art's suggested algorithm and show in detail that its complexity is indeed $O(V + E)$.

Task 7

2 Marks

Amend your C program to now also produce `task7`, a program that prints out a list of all critical servers for the original network (the network before the outage. Your C program has to implement the algorithm developed in Task 6.

The required output when run with the input files `./task7 tests/network-1.txt < tests/outage-1.txt` is:

```
The critical servers are: 0 4 10
```

Note: As before, your solution to task 7 should not affect the correctness of your solutions to earlier tasks.

Written Problems

Problem 2

4 Marks – Up to 1/2 page

Remember from the lectures that a *dictionary* is an abstract data structure where you store *(key,value)* pairs. Throughout this module, you saw a few different ways to implement a dictionary, with a range of different space and time trade-offs. Your goal in this problem is to decide on the correct data structure to implement a dictionary given a specific scenario. In each of the questions, you have to explain your decision *in a maximum of 2 sentences*.

It's important to keep in mind there might be more than one correct answer. Many real world scenarios do not have a single correct solution—this is why your argument about your chosen algorithm is important.

(a) A new piece of hardware is now available to the general public, they are called Fast Hard Drives (FHD). Unlike traditional Hard Drives, FHDs do not have any mechanical parts. Similar to a RAM memory, FHDs allow random access: any memory address can be accessed in constant time by simply using a pointer to that address. However, they also allow much larger storage space.

The University of Rapidcity is planning to use FHDs to store a database of student records. The records are stored using the Student IDs as keys and these can grow indefinitely (there is no upper bound on the ID value). The expectation is that most operations in the database will be searching and accessing records. Insertions are less critical since they are usually done in big passes during admissions and deletions never happen as the University wants to keep track of its alumni. The main requirement is that searching operations should be as fast as possible, and the University is willing to spend extra money to acquire larger FHDs if it helps to speed up searching.

Which data structure would you use in this case? Remember to explain your reasoning. (1 mark)

(b) The University of Prudentville is also planning to use FHDs to store student records. They also expect most operations in the database to be searching and accessing records. However, they do not have as much money and would prefer to keep memory costs to a minimum while still having reasonably fast search performance.

Which data structure would you use in this case? Remember to explain your reasoning. (1 mark)

(c) The University of Hackertown is another university who wants to employ FHDs for their database and they have similar requirements to Prudentville. However, they developed an internal, proprietary method that allows *parallel access to contiguous records in a FHD* using a single reading operation. They also enhanced their FHDs with a *RAM cache* that stores the result of this parallel operation and allow 100x faster search if the record is in this cache.

Which data structure would you use in this case? Remember to explain your reasoning. (1 mark)

(d) The University of Mysteryport realizes that their student record system becomes increasingly slower when newly added records are being searched. However, the FHDs show that they have still 15% of unused storage.

Which data structure is the university most likely using and why is the performance of their system quickly degrading when new items are being searched? (1 mark)

Problem 3

4 Marks – Up to 1/2 page

Below is the original pseudocode for Insertion Sort:

```
function InsertionSort($A[0..n-1]$)
    for $i \leftarrow 1$ to $n-1$ do
        $j \leftarrow i-1$
        while $j \geq 0$ and $A[j+1] < A[j]$ do
            SWAP($A[j+1], A[j]$)
            $j \leftarrow j-1$
```

A company is using Insertion Sort inside one of their products. You are a cybersecurity expert that was hired by this company to assess any security flaws with their code. After a few tries, you managed to attack their Insertion Sort code and modify it in the following way (shown in bold blue colour in the pseudocode below):

```
function InsertionSort($A[0..n-1]$)
    for $i \leftarrow 1$ to $n-1$ do
        $j \leftarrow i-1$
        while $j \geq 0$ and $\mathbf{HASH(A, j+1)} < \mathbf{HASH(A, j)}$ do
            SWAP($A[j+1], A[j]$)
            $j \leftarrow j-1$
```

In other words, instead of indexing the array as $A[j]$ and $A[j+1]$ inside the "while" condition, you now have a hash function that takes the array and a index as the arguments and return an integer. Your job is to implement specific hash functions that will cause the algorithm to malfunction in different ways.

(a) Implement a hash function that causes Insertion Sort to *keep the original array unchanged*. Explain why your solution works. (1 mark)

(b) Implement a hash function that causes Insertion Sort to *always run in the worst case complexity*, even if the resulting array does not end up getting sorted. Explain why your solution works. (1 mark)

(c) Implement a hash function that causes Insertion Sort to *sort the array in reverse*. Explain why your solution works. (2 mark)

Completing the Programming Problems

We have provided you with the skeleton code for this assignment. The provided files has the following directory structure:

```
provided_files/
  Makefile
  graph.c
  graph.h
  list.c
  list.h
  pq.c
  pq.h
  task2.c
  task3.c
  task4.c
```

```
task7.c
utils.c
utils.h
tests/
  network-1.txt
  outage-1.txt
  t2-out-1.txt
  ...
  t7-out-3.txt
  t7-out-4.txt
```

You may change any of the provided skeleton or makefile in any way, but your submitted files must contain a makefile which is able to correctly compile task2, task3, task4 and task7 with the commands `make task2`, `make task3`, `make task4` and `make task7`, respectively. In addition, your program must follow the specified output format *exactly*. Compilation and execution of your program must succeed on dimefox.

If you create new C modules then you must change the `Makefile` so that your program can be compiled on `dimefox` using the `make` command.

To run the programs you must first compile with `make` followed by one of task2, task3, task4 or task7. For all tasks, you should give the network file as an argument and can send in the outage information via standard input redirection.

For example:

```
make task2
make task3
make task4
make task7
./task2 tests/network-1.txt < tests/outage-1.txt
./task3 tests/network-1.txt < tests/outage-1.txt
./task4 tests/network-1.txt < tests/outage-1.txt
./task7 tests/network-1.txt < tests/outage-1.txt
```

The `network-X.txt` files contain the input network your program will be provided for each test case, the `outage-XX.txt` files contain the input outages your program will be provided, and the `tXX-out-X.txt` files contain the expected output. Your program must match the expected output exactly (be careful with whitespace).

Note: in this assignment, unlike Assignment 1, there will be 4 provided test cases, and 2 hidden test cases per subproblem.

Programming Problem Submission

You will submit your program via `dimefox submit`. Instructions for how to connect to the `dimefox` server can be found on the LMS.

You should copy all files required to compile and run your code to `dimefox`, this includes the `Makefile` and all `.c` and `.h` files.

It's recommended that you test your program on `dimefox` before submitting to make sure that your program compiles without warnings and runs as expected.

From the directory containing these files you should run `submit` and **list all files required to compile and run your program**. For example, to submit only the provided files we would run:

```
$ submit comp20007 a2 Makefile graph.c graph.h list.c list.h pq.c pq.h
   task2.c task3.c task4.c task7.c utils.c utils.h
```

Note that you can also list all `.c` and `.h` files in the current directory with `*.c` and `*.h`, so the following command will be equivalent:

```
$ submit comp20007 a2 Makefile *.c *.h
```

For this to work correctly you should have your Assignment 2 code in its own subdirectory (*i.e.*, the only files in the directory should be the files you wish to submit).

You must then `verify` your submission, which will provide you with the outcome of each of the tests, which will be used directly to determine the number of marks your submission will receive.

```
$ verify comp20007 a2 > a2-receipt.txt
```

To view the result of the `verify` command you must run:

```
$ less a2-receipt.txt
```

`less` is a tool which allows you to read files using the command line. You can press `Q` on your keyboard to exit the `less` view.

You can submit as many times as you would like.

Any attempt to manipulate the submission system and/or hard-code solutions to pass the specific test cases we have provided will **result in a mark of 0 for the whole assignment.**

Completing the Written Problems

You will submit your solutions to the written problems to the LMS.

Your submission should be **typed and not handwritten** and submitted as a **single .pdf file**. Pseudocode should be formatted similarly to lectures/workshops, or presented in a monospace font.

The page limit for each problem is given (half page per written problem). Submissions which go over the page limit, have too small a font size or are otherwise unreadable may be subject to a mark deduction.

Make sure you confirm that your written submission has been submitted correctly.

Mark Allocation

Daniel says: *This also needs to be updated/changed accordingly*

The total number of marks in this assignment is 20. The maximum marks for each problem are:

Problem 1 12 marks (1 mark per task for tasks 1 and 5 and 2 marks per task for tasks 2, 3, 4, 6 and 7)

Problem 2 4 marks (1 per part)

Problem 3 4 marks (1 mark for each of the first two parts and 2 marks for the remaining part)

There will be 6 test cases for each programming sub-problem, of which 4 are provided to you and 2 are hidden test cases.

The marks awarded for each programming task (tasks 2, 3, 4 and 7) will be calculated by:

$$\text{Marks} = \max\left\{2 - 0.5 \times \text{Test Cases Failed}, 0\right\}.$$

So passing 0, 1 or 2 test cases will yield a mark of 0, while passing 3, 4, 5 or 6 will yield marks of 0.5, 1, 1.5 or 2 respectively.

Late Policy

A late penalty of 20% per day (24 hour period) will be applied to submissions made after the deadline. The 20% applies to the *number of total marks*. This applies *per component*, *i.e.*,

$$\begin{aligned}\text{Grade} = \max&\Big\{\text{Programming Grade} - 0.2 \times \text{Days Late} \times 12, 0\Big\}\\ + \max&\Big\{\text{Written Submission Grade} - 0.2 \times \text{Days Late} \times 8, 0\Big\}.\end{aligned}$$

For example, if you are 2 days late with the programming component but only 1 day late with the analysis component your grade for the programming component will be reduced by $0.4 \times 12 = 4.8$ and the grade for the analysis component will be reduced by $0.2 \times 8 = 1.6$.

Academic Honesty

You may make use of code provided as part of this subject's workshops or their solutions (with proper attribution), however you **may not** use code sourced from the Internet or elsewhere. Using code from the Internet is grounds for academic misconduct.

All work is to be done on an individual basis. All submissions will be subject to automated similarity detection. Where academic misconduct is detected, all parties involved will be referred to the School of Engineering for handling under the University Discipline procedures. Please see the Subject Guide and the "Academic Integrity" section of the LMS for more information.

Appendix: Problem 1 Seen Test Cases

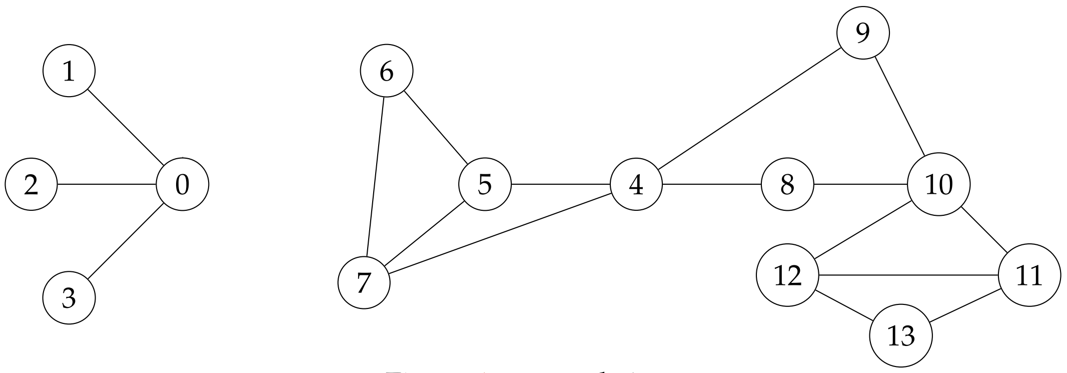

Figure 1: network-1.txt

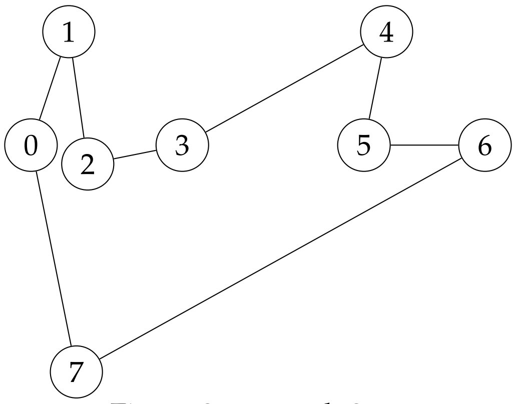

Figure 2: network-2.txt

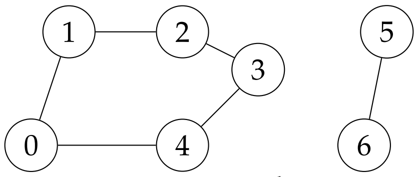

Figure 3: network-3.txt

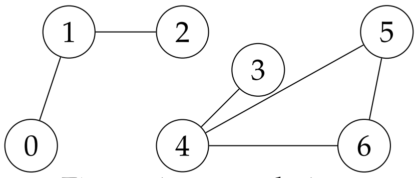

Figure 4: network-4.txt