COMP9024 25T1 Data Structures and Algorithms

Week 7 Practical
Graph Algorithms

1. (Digraphs)

   Write a program `popularityRank.c` that

   1. builds a *directed* graph from user input:

      - first, the user is prompted for the number of vertices
      - then, the user is repeatedly asked to input an edge by entering a "from" vertex followed by a "to" vertex
      - until any non-numeric character(s) are entered

      You should use `scanf` to read the input and check the return value to determine whether an `int` was successfully read.

   2. ranks all the nodes according to their *popularity*, where the popularity of a vertex *v* is defined by the following formula:

      popularityRank(v) = inDegree(v) / outDegree(v)

      (if *outDegree(v)* is zero, replace it by 0.5)

   3. outputs the nodes and their popularity ranking formatted to 1 decimal place, from most popular to least popular. Nodes that rank equally should be printed in their natural (increasing) order.

   Your program should use the Weighted Graph ADT (`WGraph.h`, `WGraph.c`) from the lecture. Do not change the header file or the ADT implementation.

   Hints:

   - You cannot use the standard Graph ADT since this assumes undirected edges.
   - You can use a modified version of the sorting algorithm (`inssort.c`) from Week 1 if you wish.

   An example of the program executing is shown below. The input is the directed graph in Exercise 2b from the Week 5 Tutorial.

```
prompt$ ./popularityRank
Enter the number of vertices: 6
Enter an edge (from): 0
Enter an edge (to): 1
Enter an edge (from): 1
Enter an edge (to): 5
Enter an edge (from): 3
Enter an edge (to): 5
Enter an edge (from): 5
Enter an edge (to): 0
Enter an edge (from): 5
Enter an edge (to): 2
Enter an edge (from): 5
Enter an edge (to): 3
Enter an edge (from): 5
Enter an edge (to): 4
Enter an edge (from): 5
Enter an edge (to): 1
Enter an edge (from): done
Done.

Popularity ranking:
1 2.0
2 2.0
4 2.0
0 1.0
3 1.0
5 0.4
prompt$
```

We have created a script that can automatically test your program. To run this test you can execute the `dryrun` *program that corresponds to this exercise. It expects to find a file named* `popularityRank.c` *in the current directory. You can use dryrun as follows:*

```
prompt$ 9024 dryrun popularityRank
```

Note: Please ensure that your output follows exactly the format shown above.

2. (Dijkstra's algorithm)

Download the implementation of a Priority Queue ADO (`PQueue.h`, `PQueue.c`). The ADO implements a single priority queue for up to 1000 elements for Dijkstra's algorithm. The queue is represented by a global variable `PQueue` of the following structure:

```
#define MAX_NODES 1000
typedef struct {
   int item[MAX_NODES];  // array of vertices currently in queue
   int length;           // #values currently stored in item[] array
} PQueueT;

static PQueueT PQueue;   // defines the Priority Queue Object
```

It is assumed that vertices are stored in the `item[]` array in the priority queue in *no* particular order. Rather, the `item[]` array acts like a set, and the priority is determined by reference to a `priority[0..nV-1]` array.

The Priority Queue ADO provides implementations of the following functions:

```
void   PQueueInit();                 // set up empty priority queue
void   joinPQueue(Vertex v);         // insert vertex v into priority queue
                                     // no effect if v is already in the queue
Vertex leavePQueue(int priority[]);  // remove the highest priority vertex from the priority queue
                                     // remember: highest priority = lowest value priority[v]
                                     // returns the removed vertex
bool   PQueueIsEmpty();              // check if the priority queue is empty
```

Next, download and complete the incomplete implementation `dijkstra.c` of Dijkstra's algorithm that uses the priority queue ADO and the Weighted Graph ADT (`WGraph.h`, `WGraph.c`) from the lecture. The program:

- prompts the user for the number of nodes in a graph,
- prompts the user for a source node,
- builds a weighted *undirected* graph from user input (adds every input edge in both directions),
- uses the priority queue ADO to compute and output the distance and a shortest path from the source to every vertex of the graph (**needs to be implemented**).

As with the previous exercise, you should use `scanf` to read the input and check the return value to determine whether an `int` was successfully read.

An example of the program executing is shown below. The input graph consists of a simple triangle along with a fourth, disconnected node.

```
prompt$ ./dijkstra
Enter the number of vertices: 4
Enter the source node: 0
Enter an edge (from): 0
Enter an edge (to): 1
Enter the weight: 42
Enter an edge (from): 0
Enter an edge (to): 2
Enter the weight: 25
Enter an edge (from): 1
Enter an edge (to): 2
Enter the weight: 14
Enter an edge (from): done
Done.
0: distance = 0, shortest path: 0
1: distance = 39, shortest path: 0-2-1
2: distance = 25, shortest path: 0-2
```

```
3: no path
prompt$
```

Note that any non-numeric data can be used to 'finish' the interaction.

To test your program you can execute the `dryrun` *program that corresponds to this exercise. It expects to find a file named* `dijkstra.c` *in the current directory. You can use dryrun as follows:*

```
prompt$ 9024 dryrun dijkstra
```

Note: Please ensure that your output follows exactly the format shown above.

Due Date

Tuesday, 8 April, 11:59:59. Late submissions will not be accepted.

Submission

You should submit your files using the following `give` command:

```
prompt$ give cs9024 week7 popularityRank.c dijkstra.c
```

Alternatively, you can select the option to "Make Submission" at the top of this page to submit directly through WebCMS3.

Important notes:

- Make sure you spell all filenames correctly.
- You can run `give` multiple times. Only your last submission will be marked.
- Where you're expected to submit multiple files, ensure they form a single submission. If you separate them across multiple submissions, each submission will replace the previous one.
- Whether you submit through the command line or WebCMS3, it is your responsibility to ensure it reports a successful submission. Failure to submit correctly will not be considered as an excuse.
- You cannot obtain marks by e-mailing your code to tutors or lecturers.

Assessment

- Each question is worth 1 mark, for a total of 2 marks.
- Submissions will be auto-marked shortly after the deadline has passed.
- It is important that the output of your program follows exactly the format of the sample output, otherwise auto-marking will result in 0 marks.
- Ensure that your program compiles on a CSE machine with the standard options, i.e. `-Wall -Werror -std=c11`. Programs that do not compile will receive 0 marks.
- Auto-marking will use test cases different to `dryrun`. (Hint: do your own testing in addition to running `dryrun`).

Collection

Once marking is complete you can collect your marked submission using the following command:

```
prompt$ 9024 classrun -collect week7
```

You can also view your marks using the following command:

```
prompt$ 9024 classrun -sturec
```

You can also collect your marked submission directly through WebCMS3 from the "Collect Submission" tab at the top of this page.

Plagiarism

Group submissions will not be allowed. Your programs must be entirely your own work. Plagiarism detection software will be used to compare all submissions pairwise (including submissions for similar assessments in previous years, if applicable) and serious penalties will be applied, including an entry on UNSW's plagiarism register.

- ***Do not copy ideas or code from others***
- ***Do not use a publicly accessible repository or allow anyone to see your code***

Please refer to the on-line sources to help you understand what plagiarism is and how it is dealt with at UNSW:

- Plagiarism and Academic Integrity
- UNSW Plagiarism Policy Statement
- UNSW Plagiarism Procedure