

## 1. (Digraphs)

Write a program `popularityRank.c` that

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

$$\text{popularityRank}(v) = \text{inDegree}(v) / \text{outDegree}(v)$$

(if  $\text{outDegree}(v)$  is zero, replace it by 0.5)

- outputs the nodes and their popularity rating rounded 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 1a from this week's problem set.

```
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
Done.

Popularity ranking:
2 2.0
4 2.0
0 1.0
1 1.0
3 1.0
5 0.5
```

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
```

## 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 ADO to compute and output the distance and a shortest path from the source to every vertex of the graph (**needs to be implemented**).

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
```

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.

## Assessment

Submit your solutions using the following **give** command:

```
prompt$ give cs9024 week5 popularityRank.c dijkstra.c
```

Make sure you spell the filenames correctly. You can run **give** multiple times. Only your last submission will be marked.

The deadline for submission is **Tuesday, 7 July 11:00:00am**.

Each program is worth 1 mark. Total marks = 2. Auto-marking will be run by the lecturer several days after the submission deadline using different test cases than `dryrun` does.

Hints:

- Programs will not be manually marked.
- It is important that the output of your program follows exactly the format shown above, otherwise auto-marking will result in 0 marks.
- **Ensure that your program compiles on a CSE machine with the standard options `-Wall -Werror -std=c11`. Programs that do not compile will receive 0 marks.**
- `dryrun` and auto-marking also check the correct usage of dynamic memory allocation and pointers as well as the absence of memory leaks.
- Do your own testing in addition to running `dryrun`.

## 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](#)

Reproducing, publishing, posting, distributing or translating this assignment is an infringement of copyright and will be referred to UNSW Conduct and Integrity for action.