DS210 Project Write-up

In a nutshell, this project returns information about a dataset based on user inputs.

Brief description of code

In order for it to be read properly, the dataset must be formatted as follows:

```
from,to
0,9206
0,7787
…
```

Here, each integer represents a Twitch user, and each line is a pair of Twitch users that are connected via following.

Using this data, the function `read_file()` returns a Graph (a struct defined in the module graph.rs, originally from lecture 28) using the implemented function `create_undirected()`.

In the main function, only two functions are called: `read_file()` so that there is data to work with, and `directory_bfs()` from the module bfs.rs. The latter function generally takes one parameter, the return value of the former (i.e. the data in Graph form), and its other three parameters exist solely for testing – specifically, simulating user inputs.

Once this function is called, the following is printed:

```
PS C:\Users\taiyo\OneDrive\Documents\GitHub\ds210_project> cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.03s
     Running `target\debug\project.exe`
Your dataset includes users ranging from 0 to 9497
Choose a user:
```

For instance, you may choose user 7923. Once you enter 7923, the following is printed:

```
Your dataset includes users ranging from 0 to 9497
Choose a user:
7923
What would you like to see?

        1: Distance to a certain other user

        2: Direct connections

        3: Mean distance to other users

        4: Exit program
```

The first three options, when selected, each call a separate function. The fourth is basically a nicer version of “Ctrl+C”.

Option 1

Let’s say you choose the first option.

```
1
Enter another user:
```

Since option 1 calls `compute_single_distance_bfs()` which finds the number of edges between two users, another user input is needed.

If you choose user 1984,

```
Enter another user:
1984
7923 and 1984 are 2 edges apart
```

you learn that 7923 is a friend of a friend of 1984.

Option 2

Let’s say you instead choose the second option.

```
2
Number of direct connections to 7923: 17
Direct connections: [114, 1490, 1920, 2097, 2845, 3034, 3370, 3663, 4038, 5894, 6242, 6690, 6918, 7275, 7586, 7715, 9327]
```

You learn that user 7923 has 17 connections, and you can also see which 17 users they are.

Option 3

If you had chosen the third option,

```
3
Mean distance from 7923 to other users: 2.7981680353758684
```

You learn that user 7923 is, on average, about 2.8 edges away from any given user.

Allowing for testing

Each time the program needs a user input, it uses the following function:

```rust
pub fn take_input(test_input: Option<&isize>) -> isize {
    let result: isize;
    if let Some(test_input) = test_input {
        //if test_input is not None, this function returns the value in test_input.
        result = *test_input;
    } else {
        //if test_input is None, this function takes a user input and returns that.
        let mut input = String::new();
        io::stdin().read_line(&mut input).expect("Could not read input.");
        result = input.trim().parse::<isize>().expect("The input must be an integer.");
    }
    result
}
```

By having an `Option` parameter, this function can simply return the content of that parameter if it is not `None`, instead of asking for a user input. This allows for testing of the program:

```rust
#[test]
fn test_good() {
    //test function for valid inputs
    let new = read_file(r"C:\Users\taiyo\OneDrive\Documents\GitHub\ds210_project\twitch\twitch\DE\musa
    //tests compute_single_distance_bfs()
    assert_eq!(directory_bfs(&new,Some(&792),Some(&1),Some(&458)), "792 and 458 are 3 edges apart");
```

By passing `Some(&isize)` parameters to `directory_bfs()`, user input can be simulated. The above example considers a scenario where a user enters 792, then chooses option 1, then enters 458. The statement

`"792 and 458 are 3 edges apart"`

should and is indeed returned.

Invalid inputs

If at any point the user inputs an integer out of bounds, a custom message is printed.

```
1st input
!(directory_bfs(&new,Some(&-92),Some(&2),None),"The user number must be within the given bounds.");
2nd input
!(directory_bfs(&new,Some(&8634),Some(&-5),None),"The input must be 1, 2, 3, or 4.");
3rd input, i.e. invalid input for compute_single_distance_bfs()
!(directory_bfs(&new,Some(&234),Some(&1),Some(&9876)),"The user number must be between 0 and 9497.");
```

Because the 2nd, 3rd, and 4th parameters of `directory_bfs()` are all defined as `Option<&isize>`, passing a string or other non-integer for any of them causes the program to not even compile. If a user does commit such a mistake, the program panics via the function `.expect()`, which is used in the aforementioned function `take_input()`.

For instance, if you get too excited and enter “hello!” instead of an integer…

```
PS C:\Users\taiyo\OneDrive\Documents\GitHub\ds210_project> cargo run
    Finished dev [unoptimized + debuginfo] target(s) in 0.02s
     Running `target\debug\project.exe`
Your dataset includes users ranging from 0 to 9497
Choose a user:
hello!
thread 'main' panicked at 'The input must be an integer.: ParseIntError { kind: InvalidDigit }', src\bfs.rs:14:48
note: run with `RUST_BACKTRACE=1` environment variable to display a backtrace
error: process didn't exit successfully: `target\debug\project.exe` (exit code: 101)
PS C:\Users\taiyo\OneDrive\Documents\GitHub\ds210_project>
```

…you are told that “`The input must be an integer`”.