Matlab Greedy Walk Project Marking Rubric

Submitted code is marked for style (is it well written?), functionality (does it work?) and speed (how long does it take to execute?).

Style marks are awarded based on the first seven specified functions (i.e. `BestPath` is excluded from the style marks). If you submit fewer than the first seven specified functions it is still possible to earn full marks in the majority of the style sections (with the exceptions being the Function names and Function Headers sections). For example, if all your submitted code is indented correctly, you would get full marks for indentation even if you only submitted a few of the specified functions.

Functionality marks are awarded for each of the eight specified functions. To get the functionality marks associated with a function your code must work EXACTLY as specified in the project document including the number, type(s) and order of any specified inputs/outputs.

The speed marks are awarded based on how fast your `BestGreedyPath` and `BestPath` functions execute, when running supplied timing scripts. The timing scripts time how long these functions take to run on the lab computers using various test arrays. The timing scripts will be released prior to the due date. To earn the timing marks you must have correct implementations of your functions (i.e. if your code executes quickly but does not give you the correct answers you will not earn the timing marks).

Style (12 marks)

Examine the submitted m files for the FIRST SEVEN functions (i.e. excluding `BestPath`) and review the following style categories. Select one of the options underneath each heading. The marks awarded for each option are shown in brackets. When peer reviewing on Aropa remember that if you mark someone down for any style element you need to include some written feedback as to why. *Note that BestPath will not be marked for style as it is anticipated most students will not manage to complete the BestPath function, you should however still follow usual style conventions for BestPath if you have submitted it.*

Function names (2)

- **As specified (2)** The first seven specified functions are named correctly, using the **exact** specified names, including the correct spelling and case (`Reverse`, `FindSmallestElevationChange`, `GreedyPick`, `GreedyWalk`, `FindPathElevationsAndCost`, `BestGreedyPathHeadingEast`, `BestGreedyPath`)
- **One incorrect filename or correct filenames but some of the first seven functions missing (1)** One of the specified first seven functions is not named correctly (remember the names must match up exactly, including case). Alternatively fewer than seven of the first seven specified functions were submitted but those submitted were still all named correctly.
- **Two or more incorrect OR one incorrect and others of the first seven missing (0)** Two or more of the specified first seven functions were not named correctly (remember the names must match up exactly, including case). If they failed to submit all of the first seven functions and one or more of their first seven submitted functions is named incorrectly, they also get zero.

Headers for functions (2)

- **Well written headers for all the first seven specified function files (2)** There is a well written, easy to understand, header for each of the first seven specified function files. The header describes the purpose of the file, including the inputs and outputs.
- **Well written headers for four to six of the first seven specified function files (1)** There is a well written, easy to understand, header which describes the purpose of each function file for between four and six files (inclusive).
- **Well written headers for three (or fewer) of the first seven specified function files (0).** Note a function may lack a well written header for the following reasons;
    - The function was not submitted
    - The function was submitted but lacks a header entirely
    - The function was submitted with a header but it is poorly written and hard to understand (or is missing important information such as a description of the inputs and outputs)

Other comments (2)

- **Well commented (2)** The comments clearly describe the purpose of the code
- **Commented but not to a great standard (1)** The comments use language which is unprofessional and/or are not sufficiently descriptive
- **None (0)** There is no commenting in the code (other than the headers)

Indentation (2)

- **Perfect (2)** All of the code is indented correctly according to the standard code conventions
- **One file incorrectly indented (1)** There is one file which contains lines of code that are not correctly indented
- **Inconsistent (0)** There are two or more files which contain lines of code are not correctly indented, according to the standard code conventions

Layout (1)

- **Code is nicely spaced out (1)** Code grouped into logical chunks, as seen in Summary Programs.
- **Poor layout (0)** No blank lines used to group codes in chunks and/or comment lines are hard to read due to being overly long

Variable names (1)

- **Well chosen (1)** All variable names either give a good indication of what the variable is used for, use a sensible abbreviation (with a comment to indicate what is stored in the variable) or follow standard conventions (e.g. using i for loop variables)
- **Poorly chosen variable names (0)** Some variable names are not easily understood and are not well commented

Code repetition (2)

- **Lines of code are not unnecessarily repeated (2)** Loops and/or functions have been used to avoid unnecessary repetition of lines of code. Variables have not been needlessly duplicated.
- **One chunk of code repeated (1)** There is one chunk of code which should have been written using a loop, that have been done without one (e.g. if four or more lines in a row are identical or similar, you should have used a loop). Alternatively there may be repetition of code from another function (e.g. cutting and pasting code from another function rather than calling the original function).
- **Very similar lines of code repeated (0)** There is more than one chunk of code which should have been written using a loop, or made use of an existing function.

Functionality (24 marks)

To test functionality, you will run a supplied master test script. The test script will test each specified function with a range of inputs and compare the outputs against the expected results. This will then generate a mark for each function. Some test scripts will be released prior to the due date.

Note that if functions have not been named correctly you may need to manually edit the name in order to be able to use the test script (remember to deduct appropriate marks in the style section if the names don't exactly match).

If a function does not take the specified input type(s) and return the specified output type(s) in the specified order it will fail the tests and they will get zero marks for that function.

The test scripts supplied before the due date will be similar but not necessarily identical to the ones that will be used to mark the project (in particular the marking script may use a range of different arrays).

Tasks 1 through 8 will all be marked out of 3 marks each. Remember that each of these eight tasks requires implementing a function. The functions are as follows.

```
Reverse, FindSmallestElevationChange, GreedyPick, GreedyWalk,
FindPathElevationsAndCost, BestGreedyPathHeadingEast, BestGreedyPath, BestPath
```

Each of the above functions will be marked using the following rubric:

Task Functionality Mark: (3)

- Implemented exactly as specified in the project document (3)
- Partially implemented, failing some tests but passing at least two thirds of them (2)
- Partially implemented, failing some tests but passing at least one third of them (1)
- It has not been implemented as specified (0)

Speed (4 marks)

As well as the marks for individual tasks, there are 4 marks associated with your overall execution time (as measured using the provided timing script, which uses tic and toc to time your code). Execution time is marked by running the timing script several times and looking at the result. Note that your code must also produce the CORRECT output to earn the timing marks (i.e. the BestGreedyPath and BestPath functions must have earned 3 marks for functionality in order for you to be eligible for the respective timing marks)

Overall performance when timing BestGreedyPath: (2)

- Test array processed in under 10 seconds (2)
- Test array processed in 10 to 20 seconds (1)
- Test array not processed in under 20 seconds (0)

Overall performance when timing BestPath: (2)

- Test array processed in under 10 seconds (2)
- Test array processed in 10 to 20 seconds (1)
- Test array not processed in under 20 seconds (0)