BME 1108A Project 4: KidneyDiDi

Program Design Due: Sunday, 16 June 2024 by 23:55 pm (Individual or Partner Submission!)
Project Due: Sunday, 23 June 2024 by 12:00 pm (Individual or Partner Submission)

Engineering intersections: Industrial & Operations Engineering / Computer Science; Healthcare Access Logistics

Implementing this project provides an opportunity to practice file I/O, functions, loops, and data structures (arrays and structs).

The program design is worth 10 points, the functionality portion of the final submission is worth 100 points, and the style and comments grade is worth 10 points.

You may work alone or with a partner. Please see the syllabus for partnership rules.

Project Roadmap

This is a big picture view of how to complete this project. Most of the pieces listed here also have a corresponding section later on in this document that goes into more detail.

1. **Read through the project specification (DO THIS FIRST)**

2. **Complete the Program Design and submit it to Moodle**
   Please do this first! I'll ask to see it before I help you with any coding or debugging questions.

3. **Write helper functions to handle the data input part of the program**
   Get these parts working first; verify you are reading in the data correctly from files!

4. **Create the location grid and determine the route**
   To debug your loops, use printf statements to print some intermediate variable values to the command prompt; you can use these values to check your approach and calculations.

   Once it's working, delete the printf statements and put in the code to print to the output file. Hint: printf and fprintf work similarly! Just a different target for the output!

5. **Check that your program is sufficiently and correctly commented**

6. **Verify that you can reproduce the Test Cases in the Appendix**

7. **Submit kidneyDiDi.c to Dr. E for testing**

Background

Our company plans to create an in-house delivery service for our new kidney implants in order to efficiently and cost-effectively supply our kidney implants at the point of care. While logistics generally fall under the purview of industrial engineers, our CEO feels that we now have enough knowledge to align our own network of delivery systems with those of existing ride-share companies, such as DiDi. Thus, we will partner with DiDi to form KidneyDiDi™!

The DiDi database has a list of coordinates corresponding to common rideshare locations in a few countries, but we will expand this system to the rest of the world for the delivery of our product. Note: other rideshare companies, such as Lyft and Uber, may provide additional information to help us in our goal, as aiding those in need of kidney transplant is good for company morale and the greater good of the human race (and also the long-term bottom line profits of the company).

Expanding DiDi's service requires a computer program that will automatically plan routes based on the ***shortest path*** through the customers' destinations. Each driver is restricted to their own range near a distribution center built by our company, so the program must limit the destinations to only those locations within a maximum area defined by a grid (hint: a 2-D grid is similar to Cartesian coordinates). Several grids will be placed around the world.

Your Job

In this project, your job is to find the shortest path from each hospital location to the next within your DiDi driver's range.

Engineering Concepts

There are two high-level engineering concepts that will be used in this project: the ***traveling salesperson problem*** and ***data corruption***. These two terms are briefly explained in the next sections.

Traveling Salesperson Problem (TSP)

The traveling salesperson problem is a commonly-used optimization problem that asks,

> *Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city?* [Wikipedia]

The "cities" and "distances" can have different definitions according to the application of the TSP algorithm. The problem has analogous questions in circuit design (shortest distance the laser needs to travel to create a circuit board – this also applies to efficient use of a laser cutter, like the one we have in the BME department!), delivery logistics (shortest time to complete package delivery to homes and/or businesses), school bus routes, and similar engineering challenges.

There are many different algorithms that have been developed to solve this problem. In this project, we will use the nearest neighbor algorithm to determine which hospital to go to next.

The nearest neighbor algorithm we will use is defined here as:

1. Start at a given location (distribution center A)
2. Find the closest location that has not already been visited
3. Go to that location
4. Track that this location has been "visited"
5. Repeat steps 2-4 until all locations have been visited
6. Go to the specified end location (distribution center B, back to distribution center A, or driver home)

A sample path found using this algorithm is shown in Fig. 2 (see the Project 4 Overview PPT to see the path animated). Note that this is a "greedy" algorithm -- it picks the single closest location that has not already been visited. It does NOT consider where the other yet-to-be-visited locations are, traffic in and out of those locations, or total distance traveled. Therefore, this greedy algorithm does not generally find the *best* path through all the required locations, but it often finds a path that is close to optimal. Step 6 above is required for our particular KidneyDiDi™ application -- it is not a necessary part of the nearest neighbor algorithm.

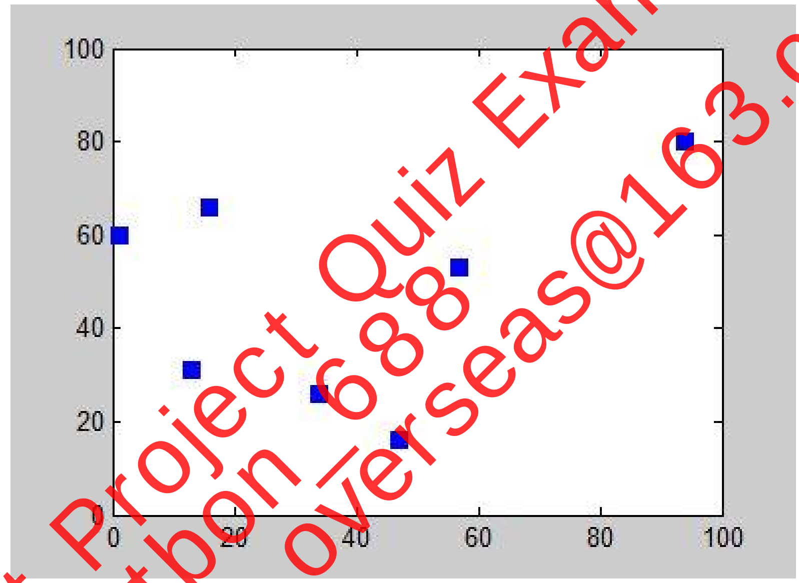

Figure 1. Example of how the nearest neighbor algorithm works. Note how the path changes based on the starting location!
(This is a .gif so to view it, see the title slide of the Project 4 Overview)

To determine which new location is "closest" to the current location, we will assume that the hospitals all lie in roughly the same plane, despite being on the curved surface of the Earth (given what you know about calculus, this should make you feel comfortable about making this assumption). Think of this as a 2-D map. Thus, we can use Cartesian coordinates and the following formula for distance:

$$distance = \sqrt{\left(x_{current} - x_{potential}\right)^2 + \left(y_{current} - y_{potential}\right)^2}$$

where $x_{current}$ and $y_{current}$ are the coordinates of the current location and $x_{potential}$ and $y_{potential}$ are the coordinates of the next possible location to travel.

Data Corruption

Transmission of data is subject to potential corruption. Data corruption occurs when data from one computer is sent to another computer, and the data received by the second computer does not match what was sent by the first computer. Data are stored in computers in binary form: a bunch of 0s and 1s. When this data gets sent from one computer to another, it is possible for a 0 to get flipped to a 1, and vice-versa, due to noise in the system. This means the data are *corrupted* and need to be fixed. There are many ways of checking for data corruption; error detection and correction is a critical component of modern computing. In this project, we will implement a simple error detection and correction based on known error effects. See below for further information.

Program Tasks

For this project, you need to determine the shortest route for your KidneyDiDi™ driver. The route starts from a given starting point (distribution center A), proceeds to the customers' hospitals, and ends at the given endpoint (another distribution center B, back to distribution center A, or to the driver's home). You will need to read in data from input files, determine the route, and create an output file with a "map" of the hospitals and the route the KidneyDiDi™ driver needs to take to get from the starting point to the endpoint. These tasks are detailed in the next sections.

Input Data

There are **two (2) input files** that are required by your program: 1) a directions file, and 2) a hospital names file. The files themselves are defined in the Input Files Section. The sections below describe the procedures (actions!) needed to process the data within the input files.

Get Filenames

Your program must prompt the user for the names of the two input files (this makes the program flexible -- the person using it can name the files whatever they want). Prompt the user for the directions file first using "**Enter Directions Filename:** "; the user then enters the <directions_file>.txt name. Then, prompt the user for the hospital names filename, <hospital_file>.txt with "**Enter Hospitals Filename:** ".

Open Files

Once ***both*** input filenames have been entered by the user, the program must then **open** the files for **reading**. If either of the input files is not found or cannot be opened for reading, the program must output "**Input file did not open properly**" to the Command Prompt, followed immediately by (on a separate line) program exit with code return 1; (note: this is using explicit return as an exit sequence since main() is a function!). Note that you do NOT need to determine ***which*** file caused the failure; simply print the message and exit the program. Refer to the lecture notes (L18A) for a reminder of how to check for errors in reading input files.

Note

The provided test cases do not test the error handling of input files. You must generate your own test case to ensure completion of this task. Do not submit your test case, just be sure your code satisfies this important project specification.

Read Data

The directions file and the hospital names file are detailed in the Input Files section. Note that the directions file contains the range for that particular KidneyDiDi™ driver (i.e., number of rows and columns in their 2-D "map"), the starting and ending points for the journey, and the location data (x,y-coordinates) for the hospitals. The names of the hospitals are stored separately in the hospital names file. Input all the directions and hospital names data, and appropriately store the information for further use (Note: you choose the storage method – Reminder: in computing, storage means use of variables to store information in the memory!).

Recommendation: use structs!
We *highly* recommend that you use an array of structs to store the hospital data! See the recent lecture notes (L20) and Final Lab Specs Review on **structs** for inspiration.

As you check each hospital in the file, you must **ensure that its coordinates lie within the grid** – if the coordinates are outside of the range, your code must output "<Hospital ID> out of range – no delivery" to the Command Prompt/Terminal, where <Hospital ID> is the ID of the Hospital (capital letter is a hint here!) whose coordinates are not within the grid.

Correct Corruption Errors

Please look at the hospital names files for the test cases. You will notice that some of the hospital names have been corrupted during file transmission: there are "OOO"s that have been erroneously added into the hospital names. You need to:

- Find and erase occurrences of "OOO" from each name
- Find and replace all of the hyphens (-) with spaces.

Create Location Grid

As part of the output file, bestRoute.txt, you must provide a grid of characters representing the "map" of locations within your KidneyDiDi™ driver's range. The directions file indicates the number of rows and columns that must appear in the grid. The directions file also contains a 1-character symbol representing that hospital.

Place each hospital's symbol at its matching grid location (row, column). If a map location is not represented by a hospital symbol, then place an underscore ("_") at that grid point. Also, place an "S" at the starting point, distribution center A, and an "E" at the ending point, distribution center B (or distribution center A or driver's home). **If the start and end point**

are the same location, mark this map location with an H for "Home". Note: for formatting purposes, also add a space after each character in the grid!

Once you have created your grid point map, output the map to file bestRoute.txt.

Watch your indexing!

On your 2-D grid point map, input row and column values start from 1, not 0. Row numbers increase from the top row to the bottom row and column numbers increase from the left column to the right column. In other words, the upper left corner of the grid map corresponds to (row 1, column 1) – similar to matrix row/column indexing in MATLAB! Note that if you store your map in a 2-D array, C array indices begin with index 0. Check, then double check your indexing!

Determine the Best Route

Acting as dispatcher for KidneyDiDi™ drivers, implement the nearest neighbor algorithm. To determine a route, begin at the start point S (row & column) provided by the required directions file. From there, find the closest hospital to the current location using the distance formula provided in the TSP algorithm (page 3). Once found, move to it – this means that your current coordinates must be updated to those of that closest hospital. **Every time you visit a new hospital: 1) *mark that hospital as Visited*; 2) update current location to that hospital; and 3) use the TSP algorithm to determine the next hospital that is nearest to your current location AND *that has not yet been Visited***. After each move, you must check all remaining unvisited hospitals to determine which is nearest to your current location. Once all hospitals have been visited, your driver travels to the end point E. Remember, if S and E are the same, place an H in the grid at this shared location.

To create the driver's route, you can make use of two (2) loops:

- The inner (nested) loop should iterate through the locations **not yet visited** to determine which hospital is the closest to the current location – only based on the nearest neighbor algorithm.

  Really Important Note

  It may be that when you are deciding on the next hospital to visit, there is more than one hospital at *exactly the same distance* from your current location. If this case is true, then you must choose the hospital with the ***greater hospital ID number***.

- The outer loop iterates while there are hospitals that have not yet been visited. It should also be the place where you output the necessary text to bestRoute.txt indicating where you are going to next. Refer the description of the bestRoute.txt file and the test cases for more details.

Program Description

In general, you need to write a C program that:

1. Reads in the directions and hospital names (input two files)
2. Processes the directions and hospital names (store for later use)
3. Determines the best route to take (nearest neighbor algorithm)
4. Outputs a file summarizing the best route (output one file)

Note: The above list can be considered to be the *start* of a program design for this project. **You will be required to submit a full program design as part of this project.** Refer to the Program Design Section for more details on what you need to submit. Refer to the Project 4 Overview PPT slides on Moodle for some inspiration on how to structure your program.

Terminology

In this program description, the following terms will be used:

- **Input Files:** Files that will be used by your program, such as a .txt file.
- **Output Files:** Files that will be produced by your program, such as a .txt file.

Required Program

There is one required program file for this project. The C file for this program must be named kidneyDiDi.c. There are no helper functions required for this project but **we *highly recommend* you create some functions to help abstract away some of the complexity of this project (that is, to break up your code, or modularize, in order to directly test small pieces of it). You can copy/paste your helper functions into a separate code file to test them one-by-one (be sure to call them from the main() in each file to test!).**

Input Files

There are 2 input data files that your code must read: 1) a file containing the directions to locations your KidneyDiDi™ driver is required to visit, and 2) a file containing the names of the hospitals with their ID numbers. These two files are defined in the next sections. Snapshots of each file can be seen in the Appendix; these input files are also located in the Project 4 Moodle folder.

Recall that the names of the input files will be entered by the user via the Command Prompt/Terminal. Refer back to Get Filenames for details of how to handle this.

Required Directions File

This file contains information about the locations your KidneyDiDi™ driver must visit. The first line in the file gives the number of rows and the number of columns, respectively, defining the limits of the grid in which your driver will deliver. Lines 2 and 3 contain the coordinates of your starting point and the ending point, in that order. Next is a list of hospital locations (row, column) to which the kidney implants must be delivered.

The data about each hospital location include row and column coordinates, a symbol representing the hospital's surcharge type (for billing), and a location ID number. **Some hospitals may have coordinates that lie outside the grid dimensions – refer back to Create Location Grid for how you should handle this!**

You are guaranteed that the input will be structured correctly and all IDs given will be unique **integer** numbers (if a leading zero is present, your program can ignore it). No tricks or error proofing needed here! You are also guaranteed that the start point and the end point are within range and that the hospitals to be visited are not at the same location as either the start point or the end point (while possible to code for this situation, this would be quite cruel to require within the scope of this project).

INPUT: <location_file>.txt

```
<Number of Grid Rows> <Number of Grid Columns>
<Starting Row> <Starting Column>
<Ending Row> <Ending Column>
<Hospital 1 Row> <Hospital 1 Column> <Hospital 1 Symbol> <Hospital 1 ID>
<Hospital 2 Row> <Hospital 2 Column> <Hospital 2 Symbol> <Hospital 2 ID>
...
```

The values in this file are explained below:

- **<Number of Grid Rows>** - the number of rows there are in your grid
- **<Number of Grid Columns>** - the number of columns there are in your grid
- **<Starting Row>** - the starting point's row number (NOT index number)
- **<Starting Column>** - the starting point's column number (NOT index number)
- **<Ending Row>** - the ending point's row number
- **<Ending Column>** - the ending point's column number
- **<Hospital # Row>** - the hospital's row number
- **<Hospital # Column>** - the hospital's column number
- **<Hospital # Symbol>** - the symbol associated with the hospital's surcharge type for billing (this will always be one character in length)
- **<Hospital # ID>** - a unique integer ID that is associated with the hospital

Hospital Names File

This file contains a list of hospital IDs and names. The file will contain the same location IDs given in <directions_file>.txt, **not necessarily in the same order and not *only* these location IDs (that is, there can be extra unused information for hospitals not on the delivery list)**, followed by the name of each location. During transmission, this file can be corrupted. It may contain some meaningless triplicates of OOO's ("OOO") which require removal. You must also replace hyphens ("-") in the names with spaces in your output file. You are guaranteed that you will have the correct number of ID and name pairs in this file that correspond to valid hospitals in the directions file.

INPUT: <hospitalnames_file>.txt

```
<Hospital 1 ID> <Hospital 1 Name>
<Hospital 2 ID> <Hospital 2 Name>
```

```
...
```

The values in this file are explained below:

- **<Hospital # ID>** - a unique ID that is associated with the hospital
- **<Hospital # Name>** - a name that is a contiguous set of characters (i.e. no spaces)

Output File

There is one output file for this program: bestRoute.txt. This file contains a 2-D grid map of the route and a list of all the hospitals that have been visited *in the correct order* based on the rules given. Below is the structure for the file:

OUTPUT: bestRoute.txt

```
<Map>


Begin at <Starting Row> <Starting Column>.
Go to <Hospital Row> <Hospital Column>. This is <Hospital Name>.
Go to <Hospital Row> <Hospital Column>. This is <Hospital Name>.
...
End at <Ending Row> <Ending Column>.
```

The values in this file are explained below (see the test cases in the Appendix for an example of bestRoute.txt) :

- **<Map>** - the grid with all the valid hospitals' symbols on it
- **<Starting Row>** - the starting point's row number
- **<Starting Column>** - the starting point's column number
- **<Hospital Name>** - a name that is a contiguous set of characters
- **<Hospital Row>** - the hospital's row number
- **<Hospital Column>** - the hospital's column number
- **<Ending Row>** - the ending point's row number
- **<Ending Column>** - the ending point's column number

Hints for a Successful Program

We highly recommend using structs to store the hospital data. Structs are a good choice because each hospital has the same set of data (name, ID, location, etc. – hint: members!) and because one naturally thinks about the hospital first and then all the data associated with that hospital (as opposed to thinking of a bunch of names, then a bunch of IDs, then a bunch of row numbers, etc.). An array of structs will conveniently store your hospital data for you to use later in your program.

Recall that when you write functions that pass in large data structures, like arrays and structs, you should use pass-by-address (arrays are automatically pass-by-address!!!). Refer back to the lecture slides on when to use pass-by-value, pass-by-address, and const pass-by-address (parameter passing decision tree).

There are many, many different ways to successfully approach this project. **I encourage you to develop a good program design before coding anything.** This can be pseudocode (a list of steps in your algorithm without any code), a flowchart, or a combination of the two. You MUST submit this program design one week in advance of the project deadline (by Sunday, 16 June)!

If you are stuck trying to get started, refer back to what we've covered in lectures and labs: char array manipulation, arrays, structs, arrays of structs, functions, etc. If still stuck, ask Dr. E for help! Hopefully, by now, you understand how important it is to feel comfortable and normal asking for help!

Submission and Grading

Project 4 has two deliverables: the program design and the kidneyDiDi.c file. Grades are broken down into 3 categories: the program design, functionality, and the style and comments.

Project 4 will be graded in three parts. First, we will evaluate the *general completeness* of your program design that is submitted to Moodle and will provide a maximum score of 10 points. Second, the functionality of your program will be evaluated for your submission. Third, we will evaluate your submission for style and commenting and will provide a maximum score of 10 points. Thus, the maximum total number of points on this part of Project 4 is 120 points (similar to Project 3). Each test case is worth the same amount. Some test cases are public (meaning you can see your output compared to the solution as well as the inputs used to test your submission) while many are private (you are simply are told pass/fail and the reason). The breakdown of points for each category is described in the next sections.

Program Design

Maximum Score: 10 points

This project requires a complex program that performs two major tasks that need to be broken down into several smaller sub-tasks. Therefore, a program design document is a required deliverable. A good program design helps to find logic errors (those are the hard ones to debug, remember!) and helps a lot when describing your program to someone. Dr. E will ask to see this program design document if you ask for help debugging your program.

Project 3 used a flowchart. You may also use pseudocode for program designs (similar to writing out an algorithm or list of key steps). You may use either of these approaches, both of these approaches, a mixture of these approaches, or something else entirely. The choice is yours. If you choose to use a flowchart at any point, you can make a flowchart using pen and paper, PowerPoint or any other programs that have the flowchart shapes built-in. If you choose to use pseudocode at any point, make sure that it is a detailed list (but not too much detail!) of steps/algorithms/abstractions (helper functions). Pseudocode is NOT code…avoid using code syntax! **IMPORTANT! Be sure you understand what you are doing with program design like flowcharts and/or pseudocode, as you will likely be required to do something similar on the Final Exam!**

Your program design should show how you are breaking down the major tasks into smaller subtasks (think about how the Program Tasks section is broken into subsections for ideas). There must be a "Start" and an "End" to the program design, with a clear control flow from start to end, but there **should not be actual code in the program design** (pseudocode is okay – describing the coding action/task in human language, NOT C programming language). For example, if you are removing an element from a vector within a loop, you have to show this in the program design but not write the actual code you would use. If you have questions, show Dr. E your program design and get feedback!

Submitting Your Program Design

Your program design will be submitted via Moodle. It will be graded for general completeness and quality, i.e. that the major steps required are documented and described and that the design is clear and easy to understand and not scribbled on the back of a napkin with a sharpie. It will *not* be graded on style, graphic design, or "prettiness," so don't spend a lot of time worrying about your color scheme!

Save your program design as a .pdf or .ppt file with Name, Partner Name (or "none"), and Date Submitted at the top. Go to Moodle and upload your file. **If you are working with a partner, you only need to submit one program design file for you both!** If you are working with a partner, be sure you both contribute to this program design file!

Note on Quality of Submission
Your program design must be a document that is of professional quality. This means text is typed or clearly written and flowcharts are done using some kind of software program (PowerPoint, Microsoft Word, etc.). **Handwritten submissions must be CLEARLY written and scanned into an electronic document (.pdf).**

Functionality

Maximum Score: 100 points

Submit your kidneyDiDi.c file to Moodle for final grading. As with previous projects, you **are highly encouraged** to submit your code to Dr. E for feedback. You will be provided with a final score (out of 100 points) as well as information on *a subset of its test cases*.

You are limited to a total of 10 submission on this project. We highly recommend that you develop tests to find scenarios where your code might not produce the correct results prior to sending it to Dr. E.

You will receive a score for the functionality portion equal to the score of your best submission. Your Moodle submission will be the code that is style graded.

Please refer to the syllabus for more information regarding partner groups.

Style and Commenting

Maximum Score: 10 points

2pts - Each submitted file has English Name, Partner English Name (or "none"), Project Name, Due Date included in a comment at the top
2pts - Comments are used appropriately to describe your code (e.g. major steps are explained)
2pts - Indenting and whitespace are appropriate (including functions are properly formatted)
2pts - Variables are named descriptively
2pts - Other factors (variable names aren't all caps, etc...)

Appendix: Test Cases

Here are two test cases for you to use in designing, writing, and testing your code.

Test Case 1

Command Prompt:

```
C:\Users\Eric\Project_4>gcc kidneyDiDi.c –o kidneyDiDi
C:\Users\Eric\Project_4>kidneyDiDi
Enter Directions Filename: dir1.txt
Enter Names Filename: names1.txt
1234 out of range — no delivery
2108 out of range — no delivery
198 out of range — no delivery
2222 out of range — no delivery

C:\Users\Eric\Project_4>
```

INPUT: *dir1.txt*

```
 5 12
 1 12
 5  1
 2 10 X 4329
12  1 T 1234
 5  2 D 4823
 1  4 D 7918
 6  3 P 2108
 4  1 X 8210
 1 11 T 5862
 3  6 X 5544
–3  3 R 0198
12  3 Q 2222
 4  4 A 4531
```

INPUT: *names1.txt*

```
2222 Johnson–Memorial
```

```
0198 Vanderbilt–University
5544 Barnes–Jewish
5862 New–York–Presbyterian/Weill–Cornell
4329 Chicago–General
1234 University–of–Pittsburgh–Medical–Center–Presby
2108 Downtown–Specialty–&–Transplant
4531 Johns–Hopkins
8210 Cedar–Sinai
4823 Children's–NYC
7918 Cornell–University
```

OUTPUT: *bestRoute.txt*

```
_ _ _ D _ _ _ _ _ _ T S
_ _ _ _ _ _ _ _ _ X _ _
_ _ _ _ _ X _ _ _ _ _ _
X _ _ A _ _ _ _ _ _ _ _
E D _ _ _ _ _ _ _ _ _ _

Begin at 1 12.
Go to 1 11. This is New York Presbyterian/Weill Cornell.
Go to 2 10. This is Chicago General.
Go to 3 6. This is Barnes Jewish.
Go to 4 4. This is Johns Hopkins.
Go to 5 2. This is Children's NYC.
Go to 4 1. This is Cedar Sinai.
Go to 1 4. This is Cornell University.
End at 5 1.
```

Test Case 2

TERMINAL:

```
C:\Users\Eric\Project_4>gcc kidneyDiDi.c –o kidneyDiDi
C:\Users\Eric\Project_4>kidneyDiDi
Enter Directions Filename: dir2.txt
Enter Names Filename: names2.txt
```

```
9999 out of range — no delivery
C:\Users\Eric\Project_4>
```

INPUT: *dir2.txt*

```
15 15
11 13
11 13
 4  4 T 6831
 3 13 P 2837
 1  9 Q 7281
 0  0 Y 9999
11 12 L 2081
11 14 O 1098
 5  3 A 1832
```

INPUT: *names2.txt*

```
2081 PuOOOnOOOing–Central
6831 GrantOOOham
1832 HuaxOOOi
7281 OOOOolong
2837 Shanghai–GenOOOeral
9999 Peking–UnOOOion–Medical–CollOOOege
1098 First–Affiliated–Hospital–SUOOOOOOOMC
```

OUTPUT: *bestRoute.txt*

```
_ _ _ _ _ _ _ _ Q _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ P _ _
_ _ _ T _ _ _ _ _ _ _ _ _ _ _
_ _ A _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

```
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

_ _ _ _ _ _ _ _ _ _ _ _ L H O _

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
```

Begin at 11 13.

Go to 11 12. This is Puning Central.

Go to 11 14. This is First Affiliated Hospital SUMC.

Go to 3 13. This is Shanghai General.

Go to 1 9. This is Oolong.

Go to 4 4. This is Grantham.

Go to 5 3. This is Huaxi.

End at 11 13.