Programming in Engineering

2014-191158510-3A

PiE C++ Final Assignment

MSM - CTW

17 July 2014

General instructions

Present your results in a report that includes a short explanation of your approach, answer the questions, what you did and a description of each plot and output file. Please do not include full source code, if required you can use code segments to illustrate your point. You can come to the staff for help. Hand in your report before 8 am August 3rd by e-mail to `a.c.dejong@utwente.nl` and `a.r.thornton@utwente.nl`, do not include executables. The report must be submitted as one file; a pdf and all plots should contain axis labels and detailed captions. Furthermore, please, submit the C++ files that you create along with the report in one compressed archive. Submit all the C++ files you need so that we can compile and run the code ourselves (state in the report which platform you developed your code on, for example Linux, Windows, Mac, ...). There is no minimum or maximum length, but all parts of the questions should be answered.

Provide your name and student number in all files/documents you send to us.

Grading: Your grade will be determined by the quality of your report, code (including comments `// we advise to write comments in English`) and your performance in the oral exam. In order to pass the class, you should have solved all of the non-starred problems on the examination sheet, commented your code to understandable standard and be able to explain your work during the oral exam. To obtain a grade eight or higher, the code should be efficient, in good style and very well commented, all single-starred questions must be answered and the report should be very clear. For the top grades, some extra effort beyond the requested should be undertaken i.e. go slightly beyond what is asked (open to you) and all the double starred questions must be completed.

Oral Exam: After having received your submission, we will check it and invite you for an Oral Exam. For this please bring the report and your laptop with the source-files, so that we can ask you about the code. Also make sure your code compiles on your laptop.

Exercise 1: Computing prime numbers

Background

A prime number is defined as 'a positive integer $p > 1$ that has no positive integer divisors other than 1 and p itself.'

The brute force, but simple, method to decide if p is prime is to test if ($n \bmod q \neq 0$) for all numbers $1 < q < p$.

A quicker way to find primes is to only perform the division test for *prime numbers* less than p.

For example, to test the number 11 you only need to check if (11 mod 2), (11 mod 3), (11 mod 5) and (11 mod 7) are zero.

Questions

1. Write a C++ program to compute the first N prime numbers, where N is given by the user. Use dynamic arrays to store the primes and use this information in the mod test.

2. Write to the screen a list of the first 10000 primes in the format below; where $p(n)$ is the n^{th} prime number. Report only the last five lines. Comment on the behaviour of the ratio $n * \ln(p(n))/p(n)$ as n gets large.

```
n   :  p(n)  : n*ln( p(n) )/p(n)
1   :   2    : 0.34657359027997
2   :   3    : 0.73240819244541
3   :   5    : 0.96566274746046
```

3. Based on question 2, give an estimate of the 10^6-th prime number.

4. Instead of writing to the screen, write to a file (on disk) a list containing just the prime numbers. Print eight numbers per line, such that all numbers have the same space, i.e., the file should start like:

```
2  3  5  7  11  ...
```

*5. Time your code for $N = 10^3, 10^4, 10^5$ and 10^6. Make a log-log plot of run-time against N for both codes. What can we say from the log-log plot? Do this analysis for brute force and suggested speed up and comment on the results.

**6. More efficient ways of computing prime numbers exist. Find and implement one and report the analysis of part 5 for this algorithm. Comment on the results; think of computing very large prime numbers. Report the time your program requires to calculate the first 10^9 primes.

Exercise 2: Reverse Polish Notation

Background

During the 1950's and 1960's, computer scientists were looking for ways to program early electronic computers in an easier language than machine code (C and C++ were not yet developed). In the process, the Reverse Polish Notation (RPN) was invented, and reinvented, because it is both relatively easy to implement and convenient to work with. We normally use infixnotation for writing down calculations: the operator is positioned between the two operands, as in

```
10 + 2
```

Reverse Polish Notation uses the postfix-notation, writing the same equation as

```
10 2 +
```

The advantage of this system is that there is no need for an order to be defined for operations; while the equation

```
(10 + 3) * 7
```

needs brackets in the infix-notation, the same equation reduces to

```
10 3 + 7 *
```

in RPN.

Questions

1. Read a string input from the terminal (which is assumed to be in RPN). Interpret the string correctly and output the result to the screen.
   Your Reverse Polish Notation calculator should be able to do add, subtract, multiply and divide integers.

2. Extend your code such that it reads the input line-by-line from a file. Each newline marks the end of each calculation. Write out the result of each line of the calculation and your program should abort when it detects an 'end of file'condition.

*3. Add in error checking to confirm the entered string can be interpreted as RPN string.

*4. Add in support for the power operator using the symbol ^.

**5. Extend your code so you can read from and write to variables.

```
3 1 + =TEST
```

should create a variable called 'TEST' and set it's value to 4.

```
?TEST 5 -
```

should print -1 if TEST exists and is equal to 4.

Exercise 3: TomTom

Background

As the name suggests the aim of this problem is given a set of roads connecting points (cities in our example for simplicity) find the shortest route between any given two cities. Hence the TomTom problem.

Questions

Assume a network of L roads connecting N cities; each road connects two of these cities. You are given a text file `distances.txt` that contains a double array of the length of roads connecting the two cities. The i^{th} row, j^{th} column gives the travel time from the i^{th} to the j^{th} city. If there is no direct connection between two cities the array contains a ridiculously large number, 123456789; on the diagonal the number 0 is placed. For example if we have the following matrix

$$\begin{array}{ccc} 0 & 123456789 & 100 \\ 123456789 & 0 & 400 \\ 100 & 400 & 0 \end{array}$$

This implies the first city has no connection to the second but is a distance of 100 from the third. Whereas the third city is connection to the second and is a distance of 400 away.

You are given several different sized sets of cities for each:

1. For each set of cities compute the shortest distance between every set of cities and write this information to disk with the route as a list of cities. Use the format cities, route, distance i.e. for our example above the answer would be

$1-2$	$1-3-2$	500
$1-3$	$1-3$	100
$2-3$	$2-3$	400

Use any algorithm you see fit.

*2. A famous Dutch mathematician and computer scientist (Edsger Dijkstra) discovered an efficient algorithm for finding the shortest route between two points. Even though it was first published in 1959, it remains the fastest algorithm to actually give the shortest route.

Dijkstra's algorithm
First, to initialise:

(a) Set visited for all cities to false.

(b) Set distance for all cities to infinite.

(c) Set distance for the first city to 0.

Next, to find the shortest route:

(a) Find the city with the lowest distance which has not been visited yet.

(b) Mark the city as 'visited'.

(c) If the city is the endpoint, stop.

(d) Update the connected cities with the distance found.

Construct the actual route

(a) Start at the last city.

(b) Look which connected cities has the lowest distance.

(c) Add that one to the route.

(d) Consider the city just added to be the last one.

(e) Is the last city the begin city? Done!

Implement Dijkstra's algorithm.

Hint: Write a class 'City', which stores the best distance so far, the position, the name and the other cities which it connects to.

**3. The real TomTom problem. The real Dutch road network has 825000 road junctions just counting the motorway and the provincial roads:

(a) Use some means to estimate how long your code from both part 1 and part 2 would take to compute a route between two randomly chosen junctions on the Dutch principle road network.

(b) Your TomTom has a very low powered processor and has to give a route within 15 seconds. It is no use if it would take two hours to compute the route. Think carefully about what it must do to achieve this (hint, TomTom and Garmin will not always give the same route for the same input). Implement an algorithm that can deal with the real Dutch city file in under 30 second on your laptop (note, only compute one example route). This is still easier than what TomTom must do.

(c) You are now given an easier problem. For the same set of cities you can pre-compute what ever data you want, up-to a limit of 1 Megabyte. Precompute some data and use this precomputed data to improve your algorithm from part (b).

Because the matrix format as given for parts 1 and 2 of the exercise can no longer function for any real-size map, a different data format is used. The data format lists vertices, roads and cities. Cities are only required to find start and endpoint, as for instance the street 'Hengelosestraat' can exist in several locations. The following data format can be expected:

```
number_of_cities number_of_vertices number_of_roads
city_index0 x y Name
city_index1 x y Name
...
city_indexN x y Name
vertex_index0 x y
vertex_index1 x y
...
vertex_indexN x y
road_index0 type maxSpeed N v1 v2 v3 v4 ... vN Name
road_index1 type maxSpeed N v1 v2 v3 v4 ... vN Name
...
road_indexN type maxSpeed N v1 v2 v3 v4 ... vN Name
```

Every city lists an ID, coordinates and a name. Every vertex lists an ID and coordinates Eery road lists an ID, a type, a maximum speed (in km/h), a number of vertices, the vertices and a name of the road. The 'type' field can contain any of the followin values:

A Highway

N National road

S Normal street

R Residential road

For example, the following file describes 3 cities, 4 points and 2 roads between those points.

```
3 4 2
0 52.2197034 6.8809074 Enschede
1 52.2571266 6.7991122 Hengelo
2 52.2488822 6.906319 Lonneker
0 52.2197034 6.8809074
1 52.2571266 6.7991122
2 52.2488822 6.906319
3 52.24036 6.91512
0 N 80 3 0 3 2 N732/Oldenzaalsestraat
1 S 50 2 0 1 Hengelosestraat
```

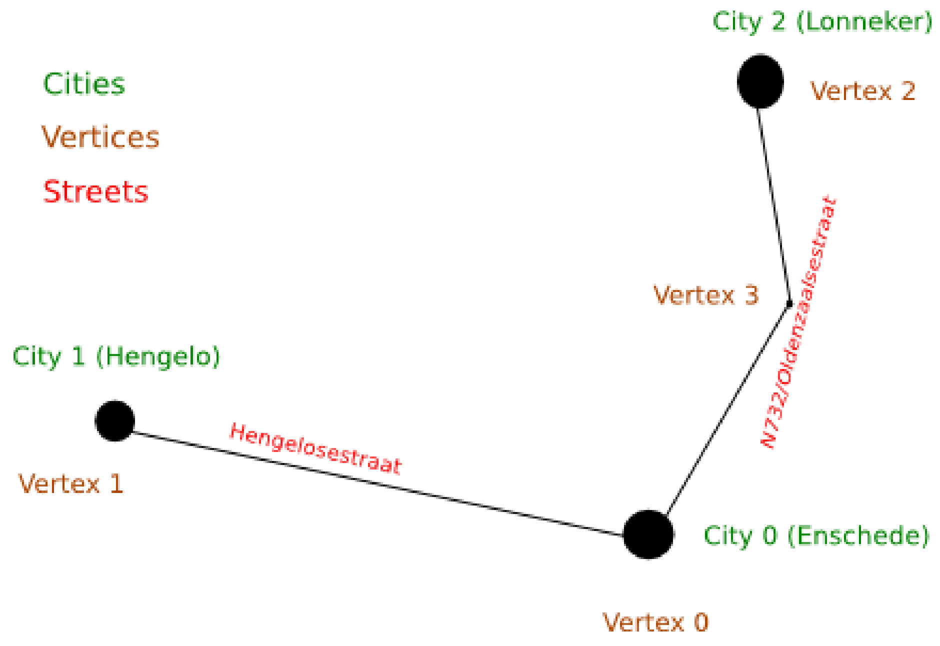

Figure 1: The same map, drawn