UNIVERSITY OF CAMBRIDGE
Computer Laboratory

Computer Science Tripos Part IB
Group Project

REQUIREMENTS SPECIFICATION

Charlie
the Canal Routing Program

Group Charlie

Nathan Dimmock
Martin Harper
Karen Inglis
Phebe Mann
Andy Oakley
David Stimpson

February 2000

Amendment to Requirements Specification

Amendment Rev. No.	Date	Amendment	Remarks
1	2/2/2000	Expression amended	

Amendment to **3.3.2** Section on **Edges**

Two vertices are adjacent if:

$$((|(v.x - u.x)| \leq 1) \wedge (|(v.y - u.y)| \leq 1) \wedge ((v.y \neq u.y) \wedge (v.y \neq u.y)))$$

The expression is amended to :

Two vertices are adjacent if:

$$((|(v.x - u.x)| \leq 1) \wedge (|(v.y - u.y)| \leq 1) \wedge ((v.x \neq u.x) \vee (v.y \neq u.y)))$$

CHARLIE

Canal Routing Program

REQUIREMENTS SPECIFICATION

CONTENTS

1. Introduction

1.1 Background

The cost of building a canal is highly dependent on the terrain it travels through and as such the most direct route from A to B is not always the cheapest to build, or the quickest for boats to traverse. Finding the most economical route involves time−consuming and repetitive calculations, ideal for automation by a computer system.

1.2 Investigation

At our first meeting we decided that we would interpret "most economical" as referring to building costs and ignore all other factors such as whether a canal route would be commercially viable or would be underused if built, because of the route it took. We also determined that finding the optimum route between two points would involve a graph−based algorithm. We therefore looked in textbooks (in particular, Introduction to Algorithms by Cormen, Leiserson and Rivest) for graph algorithms that could be adapted to suit our problem. The problem we are trying to solve is a single−pair, shortest−path problem. However, no algorithms for this problem are known that run asymptotically faster than the best single−source, shortest−path algorithms in the worst case, so we were able to reduce our problem to finding the most suitable single−source, shortest−path algorithm.

We thought that it would be highly improbable for someone to gain money by building a canal, and therefore decided to disallow negative costs for the construction of the canal. This meant that the algorithm we chose did not have to be able to cope with negative weighted edges.

Dijkstra's algorithm seemed the most suitable choice, since it allows the search for shortest paths to be terminated once the shortest path between the two desired points has been found. Dijkstra is also as efficient as any other algorithm, being O (V^2) where V is the number of vertices in the graph.

2. Facilities

2.1 Terrain Generator

1. Generation of a random terrain based on some restrictions given by the user.
2. Ability to save the terrain generated.

2.2 Route Finding Algorithm

1. The user will supply a terrain map and the cost of building the different parts of the canal.
2. The program will then attempt to find the most economical route from point O to point C, as specified by the user.
3. The user will then be able to export this route to a text file or obtain a printout.

2.3 User Interface to Route Finder

1. A graphical display of the terrain.
2. A facility to allow the user to enter the start and end points.

3. A facility to allow the user to enter the costs of different canal structures (e.g. tunnels, locks, etc.)
4. A graphical display on the terrain of the route found.
5. Calculation and display of the cost of building the canal along the proposed route.
6. Ability to display the height of a square of terrain and the grid coordinates of a square.
7. Ability to save routes to a file and load previously found routes.

3. Planning of Major Components

3.1 File Formats

The Terrain

Each row of the matrix will be represented by one line in the local file encoding format. Data values in a row will be separated by commas. After the last value there will be a new line character ('\n') but no comma. The whole file will be terminated by an EOF marker.

The file should be in the local file format (UNIX/Windows) for the OS the Charlie will be running under. This should make little difference when our program generates terrain, but could be crucial when editing terrains by hand or transferring terrain generated under one OS to another.
The first row will set the number of columns, if there is a different number on any subsequent line, Charlie will inform the user that the file format is invalid and not load the terrain

The data values will be integers representing height above sea level, so negative numbers indicate distance below sea level. A number followed by an X char indicates that a square contains some impassable terrain for the canal, such as a village, SSSI, etc. It is not possible to tunnel under impassable squares.
An example

```
27,10,34,35,17,12,8\n
28,12X,36,32,15,18,−5X\n
EOF
```

The Route

Charlie will have a binary (non−readable) output using the serialisable interface to allow the route to be loaded and saved. It will also have an output in a readable format, which can be used for exporting and printing the route. The readable format is described below.

- The file will a text file in the local encoding.
- The first line of the file contains the name of the terrain map. The filename should be relative so that it can cross platforms. The default file extension will be .ter for terrains and .rte for routes.
- Each subsequent line contains co−ordinates of one vertex that canal passes through, and the height of the canal relative to the terrain, with the information separated by spaces.
- Vertices are stored in order they are visited. An example printout is shown below

Example

```
(0,0)(50,50)          Start as Embankment       Height 27m  45DEG−N
```

(1,1)(0,0)	Lock up (10m)	Height 9m	−
(1,1)(0,0)	Change to Standard	Height 9m	45DEG−N
(1,1)(50,50)		Change Direction	Height 9m 90DEG−N
(2,1)(0,50)	Change to Cutting	Height –23m	90DEG−N
(2,1)(50,50)		Change Direction	Height –23m 45DEG−N
(3,2)(0,0)	Change to Tunnel	Height –35m	45DEG−N
(3,2)(50,50)		Change Direction	Height –35m 0DEG−N
(3,3)(50,0)	Lock down (10m)	Height 35m	−
(3,3)(50,0)	Lock down (10m)	Height 25m	−
(3,3)(50,0)	Change to Embankment	Height 25m	0DEG−N
(3,3)(50,50)		End Height 25m	−

3.2 Terrain Generation Algorithm Specification

3.2.1 Introduction

This algorithm specification lays out the bare minimum that will be provided in terms of terrain generation. It is likely that this algorithm will be enhanced, or even completely re−developed, but that it is guaranteed that an algorithm at least as good[1] will be used.

The following parameters, as a minimum, will be supplied by the user to the algorithm:

Max height (m)
Min height (m)
Dimension (tile x tile)
Random generator seed
Corner Seeds[2] (m)

3.2.2 Notes on the algorithm

The algorithm uses midpoint displacement in two dimensions[3], and thus requires a square working grid, of side n, where:

$n = 2^k + 1 \qquad | \; k \in \mathbf{N}$

Let the tiles be labelled $t_{i,j}$ where $0 \leq i, j \leq 2^k$, it is unimportant from which corner of the grid these tiles are labelled.

The algorithm is recursive[4], and works on sub grids of the main terrain grid. Each sub grid has dimensions as described above..

3.2.3 Diamond−Square Algorithm

Initially: $res = 2^k + 1$
$t_{0,0}$ = corner seed 1
$t_{0,2^k}$ = corner seed 2
$t_{2^k,0}$ = corner seed 3

[1] How good a terrain algorithm is, will be determined by eye
[2] Defined later
[3] An algorithm that generates a fractal, with terrain−like properties
[4] Unsurprising[3]

$t_{2^k,2^k}$ = corner seed 4

The following algorithm recurses k times before termination.

Consider a tessellation of grid squares, dimension of res x res, with an overlap of 1 tile on each side over surrounding tiles. For each square perform the following:

W.l.o.g. let us take the square containing $t_{0,0}$, for other squares use the same relative tiles within the sub grid. Set the value of $t_{k,k}$ where k = (res – 1)/2, to the following:

$t_{k,k}$ = ave + rand

where:

ave = ($t_{0,0}$ + $t_{0,2^k}$ + $t_{2^k,0}$ + $t_{2^k,2^k}$) / 4
rand = 2*res – 1

Once this operation has been performed on each square, a further operation is performed on four of the tiles within each square:

Again, w.l.o.g. let us take the square containing $t_{0,0}$, for other square use the same relative tiles within the sub grid. Set the value of $t_{0,k}$, $t_{2k,k}$, $t_{k,2k}$, $t_{k,0}$ where k is defined as above, as follows:

$t_{j,j}$ = ave + rand

where:

ave is defined as the sum of each tile that is obtained by incrementing or decrementing that tile’s x or y co−ordinate only, by k, then by dividing by the number of such tiles. There will typically be four such tiles, but there may be less if the square is near the edge of the grid.
rand = 2*res – 1

Once this, further, operation has been performed on each square, this iteration is complete. The next iteration, if run, will use a new value of res, res’ = ((res –1)/2) + 1.

The resulting grid, a random height map based only on the input conditions[5], is now scaled using the max and min height inputs linearly, before returning

3.3 Route Finding Algorithm Specification

3.3.1 Simplifications

The algorithm requires the following simplifications to be made:

1. Restrict the directions that the canal can take to just 8 points of the compass – N, S, E, W, NE, SE, SW, and NW. This has to be done to make the number of vertices in the graph finite.
2. Locks can only be built at the edges of squares – i.e. canal is at a constant height within any one square. Allowing locks anywhere other than at the edges of squares would not improve the cost of the route found, since each tile is flat.

[5] Excepting the max and min height

3. The canal must pass through the centre of each square that it enters. This is a necessary condition in order to ensure that the cost of going from one square to the next is independent of the previous path taken to that point.
4. No changes of direction are allowed between the centre of one square and the centre of the next.
5. Terrain heights are to the nearest metre.
6. Every lock changes the height of the canal by exactly 10 metres. This is to ensure that there are a finite number of vertices in the graph used by the algorithm.
7. The lowest point on the terrain map is used as a base−line from which other heights can be calculated.
8. The grid size and the maximum height difference between tiles of the terrain are not restricted.

3.3.2 Algorithm

This specification lays out the base algorithm for the routing. Although it may be enhanced, or completely redeveloped at a later stage, an algorithm that produces output at least as good[6], is guaranteed to be used.

Consider a graph G. G is intended to represent a basic structure for developing the route of the canal through the terrain.

Vertices

Firstly let us consider a base height, *b* that is defined to be 30m below the terrain height of the lowest tile of terrain. The following vertices are added for each tile on the terrain map:

Consider a single tile. A vertex is added at a height *h* iff: (where *t* is defined to be the height of the current tile)

$(h - b) \bmod 10 = 0$

$h \geq b$

$h \leq (t + 30)$

These vertices represent all points through which the canal could potentially be built.

The classification of these vertices is as follows: (dependant on the vertex height *h* and the tile's height *t* such that d is defined as (h − t))

$d < -30$	Tunnel
$-30 \leq d < -10$	Cutting
$-10 \leq d \leq 10$	Surface
$10 < d \leq 30$	Embankment

Two further special vertices are added and labelled *O* and *C*. *O* and *C* are attached to all possible start and end points of the canal, respectively.

Edges

Let the sets of vertices *T*, *C*, *S* and *E* refer to all vertices that are, respectively, classified as Tunnel, Cutting, Surface and Embankment vertices. If *v* is a vertex let *v.h* represent its height. Let *v.x* and *v.y* represent the x co−ordinate and y co−ordinate of vertex *v*,

[6] I.e. produces a route of the same cost, or cheaper.

respectively. Let *v.th* represent the terrain height of the tile with the same x and y co−ordinates as *v*.

Consider two vertices *v* and *u*. *v* and *u* will be considered to be connected by an edge iff the following holds although, importantly, edges are not added to the graph at this stage.

Two vertices are adjacent if:

$$(|(v.x - u.x)| \leq 1) \wedge (|(v.y - u.y)| \leq 1) \wedge ((v.x \neq u.x) \vee (v.y \neq u.y)))$$

Two vertices can be connected by an edge if they are adjacent and:

$$(v, u \in C \cup S \cup E) \vee ((v, u \in T) \wedge (v.h = u.h)) \vee ((v \in T) \wedge (u \in C \cup S \cup E) \wedge ((u.th - v.h) \leq 30)) \vee ((u \in T) \wedge (v \in C \cup S \cup E) \wedge ((v.th - u.h) \leq 30))$$

i.e if

- neither vertex is a tunnel, or
- both vertices are tunnels, and the vertices are at the same height, or
- one vertex is a tunnel and the other is not, and the terrain height of the non−tunnel vertex tile is not more than 30 above the tunnel height.

The edge is weighted according to the cost of a straight line path connecting the two points on the terrain map, taking into account canal height, locks and tunnelling costs where applicable.

Some supplementary edges are considered, from *O* to any vertex *v* such that ($v \notin T \wedge v.x = O.x \wedge v.y = O.y$), and from C to any vertex *v* such that ($v \notin T \wedge v.x = C.x \wedge v.y = C.y$). The edges should hold a weight of 0.

Route Finding

The cost of the path that directly [7]connects *O* and *C* (regardless of the terrain) is determined, let that cost be *m*.

A modified Dijkstra’s algorithm [8]is then run over the graph *G*, taking *O* as the start point. The modifications are as follows:

Since the graph initially has no stored edges, it will generate and store each edge considered as the algorithm progresses.

Each vertex in the graph for which the least paths have been found, stores some pointer to the previous vertex in its least path.

Any vertex, for which the cost of the most economic path has been determined, that has a cost of strictly greater than m, is deleted from the graph.

The most economical path will then be generated by working backwards along the linked list beginning at *C*.

3.3.3 Coster Algorithm Specification

[7] As direct as possible, while still moving only from one tile centre to the centre of an adjacent tile. It need not be the most direct route, one that is sufficiently direct will suffice.

[8] Introduction to Algorithms – Cormen, Leiserson, Rivest – S25.2 − P527

In order to determine the cost of an arbitrary path throughout the terrain the process is split into individual steps. Clearly this algorithm will only suffice for paths that more from the centre of one tile to the centre of another[9], so will need to be updated should a new route generation algorithm be used.

Consider the total, running, cost to be *totcost* which will initially be set to 0. If the path traverses between *n* tile centres, then let the tiles be labelled t_o .. t_n in the order that they are visited by the path, and the vertices v_0 .. v_n. Let *k*, the value determining which edge the algorithm is currently considering, i.e. the one connecting t_k and t_{k+1}, be initially set to 0. Let the following be defined for both tiles and vertices:

x	Absolute x co−ordinate of the tile/(vertex's tile)
y	Absolute y co−ordinate of the tile/(vertex's tile)
ah	Absolute height
rh	Relative height (in case of vertex only – undefined for tile)

These properties are specified by the following, in the case of a tile, t_j.<*property*>.

Let the following iterate (n−1) times and for each iteration *k* should be set to one greater than the last iteration:

To determine the length of the canal route, in each tile, along this edge, consider that *dist* determines half the distance between the two centres. There will therefore, clearly be a canal *dist* long in each tile. *dist* is defined as follows:

$$dist = \sqrt{((v_{k+1}.x - v_k.x)^2 + (v_{k+1}.y - v_k.y)^2)}/2$$

The height ranges for the different canal types, and their respective costs (per m) are detailed as below:

$v_j.rh < -30$	Tunnel	s
$-30 \le v_j.rh < -10$	Cutting	r
$-10 \le v_j.rh \le 10$	Standard	p
$10 < v_j.rh \le 30$	Embankment	q

Let $v_j.c$ specify either of {p, q, r, s}, dependant the range in which $v_j.rh$ lies.

Let *locks* specify the number of locks required at the tile boundary (at the midpoint of the edge). *locks* should be defined to the following value[10]:

$$locks = (v_k.ah - v_{k+1}.ah)/10$$

Let $totcost = totcost + dist*(v_k.c + v_{k+1}.c) + locks$

The next iteration begins[11].

Once this iteration is complete, the value of *totcost* is returned.

[9] However, the tile concerned may actually be a fraction of the size of a full terrain tile
[10] $(v_k.ah - v_{k+1}.ah) \bmod 10 = 0$ will be true for all v_j
[11] If appropriate

3.4 Graphical User Interface

3.4.1 Specification

Specification for Graphical Display of the Route & Terrain

1. The terrain will be represented as a 2−dimensional grid of squares, with one square representing one kilometre square of terrain.
2. Different colours will be used to indicate different heights of terrain. The colours used will be similar to those used on Physical Ordnance Survey maps, i.e. a smooth variation of shades of green, brown and white.
3. The canal route calculated by the system will be displayed on the terrain grid.
4. The different types of canal construction will be distinguished by using different symbols to represent each type of construction. A key to the symbols will be provided.
5. It will be possible to zoom in on parts of the display.
6. It will be possible to scroll the display.

Information about the Route & Terrain

1. The user will be able to obtain the following information about a square of terrain:
   · The square's grid number. The square will be given a grid reference with the most SW square (i.e. the bottom left corner) as (0,0).
   · The terrain height of the square.
2. The user will be able to find out the type of construction (e.g. embankment or cutting etc.) of any piece of canal.
3. The system will calculate and display the total cost of building the canal along the proposed route. The cost will be displayed in the same currency that the initial costs were specified in by the user.

Specification for Details provided by the user

1. The user must specify the start and end points of the canal by clicking on the two relevant squares of the grid. The user will be able to edit their choice of start and end points. The route can be recalculated based on the new start and end points, but if the previous route was not saved then it will be overwritten. The user will be warned if they are about to overwrite an unsaved route.
2. The user must specify the construction costs of the canal. These costs must not be negative. The costs may be in any currency. As with the start and end points, the user will be able to edit the costs. The costs that the user is required to specify are as follows:
   · Surface Canal (per metre)
   · Cutting (per metre)
   · Embankment (per metre)
   · Tunnel (per metre)
   · Lock (per lock)
3. The graphical display will incorporate a way for the user to indicate that they have finished altering the costs, and start and end points and wish the route to be calculated.
4. Invalid input will be reported to the user.

Input & Output

1. It will be possible to load and save terrains along with the routes found. The cost information for the canal construction will also be saved with the route.
2. In order to allow the route to be accurately detailed there will be an additional grid reference system within each square. The sub−grid will range from (0,0) in the bottom left corner of the square, to (100,100) in the top right corner. Each square tile will contain its own grid of this sort. For example, (0,5) (100, 100) would be the top right hand corner of the terrain square with grid reference (0,5).
3. The route will be saved in a human readable format, which can then be printed.

Access

1. The Graphical Display will use the cross platform Look and Feel (Metal).
2. The Graphical Display will be easily useable in 800x600 and larger screens. It will be useable on 640x480 screens, but only when maximised.
3. The Graphical Display will rely on Java™ 1.2 and Swing™, and will work on any platform supporting these standards.
4. Popup help will be used where applicable, to complement the documentation.

Possible Optional Extras

1. Converting the display from 2−D to 3−D.
2. A grid over the terrain map to indicate the edge of each of the squares of terrain, along with a facility to allow the user to remove the grid.
3. Facilities to allow the user to find out the following information:
   · The exact height of the canal above a square of terrain.
   · The total length of the canal.
   · The total number of locks used along the route of the canal.

4. Acceptance Criteria

4.1 Testing

The system will provide all of the facilities listed in the Facilities section of the Specification. We will ensure that all of the facilities work correctly and the system conforms to the specification by carrying out the following rigorous tests:

1. Each of the major objects (classes) of the system will be tested in isolation, by supplying test data to it and observing the output directly, rather than via another module.
2. A terrain generator will be written that will allow us to produce a large number of pseudo−random terrains which the algorithm can be tested on. The terrain generator will allow us to limit certain parameters of the generated terrain to help us ensure a wide variety of test cases. Each terrain will be saved and so can be easily reused.
3. We will generate particular terrains by hand to allow us to test special cases.
4. For each test map, we will compare the route found by the algorithm with one generated manually, in order to ensure that the algorithm is working correctly.

5. We will ensure that the Graphical User Interface works correctly by testing it exhaustively on a wide range of people who are not members of the project.

4.2 Testing of Limitation

We will attempt to find the limitations of our program by:

1. Testing on a wide variety of machines and Operating Systems although our aim is for the system to run at an acceptable speed when supplied with sensible and realistic data on a typical desktop PC (e.g. a PII 450Mhz, 128Mb RAM, Windows 95 & Linux).
2. Test the system with maps of a maximum size of approximately the area of North America, with mountains a maximum height of Mt Everest, and a maximum gradient of 500m/km.

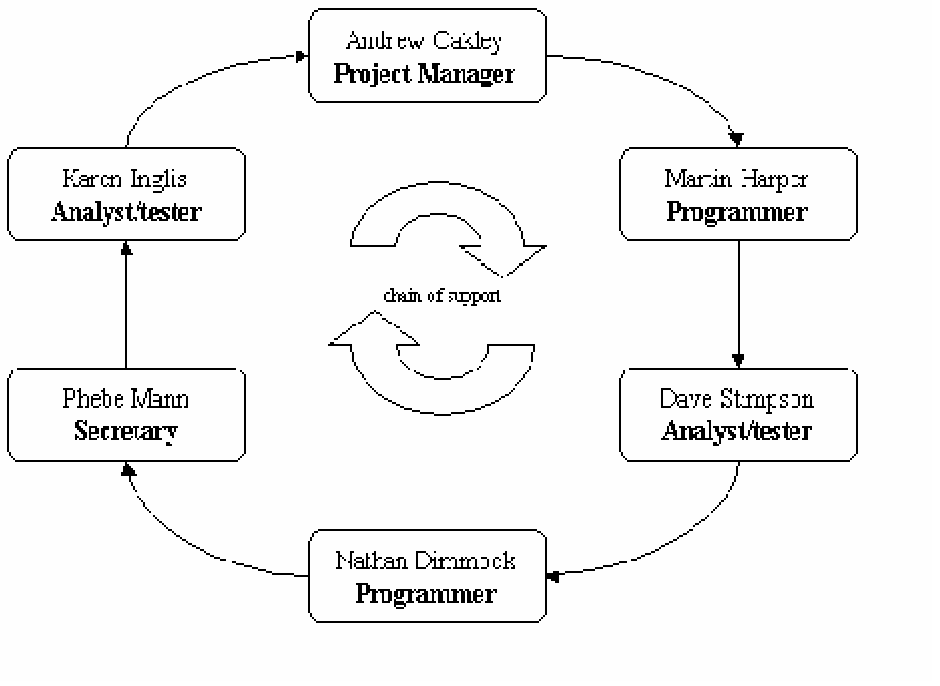
5. Management Strategy

5.1 Objectives

From a management point of view, the project takes on quite a different perspective. Given the time allocation and the size of the task, it becomes clear that working together efficiently as a team will become paramount. Working to meet the deadlines is also an important part of the project and will require coordination and communication between all members. Time sheets are kept for the purpose of budgetary control.

5.2 Group Organisation

The team structure was discussed and decided upon at the Inaugural Meeting. The Organisation Chart below shows the assignment of positions to each team member.



Note :Project Manager takes also the role as Code Librarian and Secretary takes also the role as Documentation Writer.

5.3 Resources and tools

Two systems are currently being used for the project. CVS 1.10.7 is used on a server in Jesus College to maintain the consistency of the shared workspace whilst allowing each team member to keep an entire copy on their personal machine. This repository is copied an hourly basis to Thor which also provides the server for the group website. Images of the CVS repository are also being committed to CD−ROM on daily basis for backup purposes.

It has been agreed that the Java 1.2 specification will be used for all programming related material. All internal documents and the website are stored in HTML 4.0 format so as they may be viewed on a variety of platforms.

The user manual is will be provided in HTML 4.0 and hardcopy.

All source code is to be commented in the Javadoc style. This enforces consistency throughout the code and allows HTML module descriptions to be generated with relative ease.

5.4 Communication

At the first meeting, it was agreed that regular ’conference’ e−mails would be sent to keep everyone informed of developments and resolve any disputes arising. In cases of important decisions or emergencies, contact by telephone has also been arranged.

In preference to sending attachments by e−mail, it was decided that the website should be used to publish all documents (including source code) currently being worked on. With CVS in place, this allows any team member to modify these documents adding comments where they feel appropriate.

Regular group meetings are arranged twice a week in additional to the scheduled review meetings with our group supervisor. These provide a medium for resolving problems, making crucial decisions and updating progress.

5.5 Project planning

The plan below outlines the provisional tasks for each team member until the final review meeting. The rightmost column corresponds to the fortnightly review meetings.