



INTERGEN CODE TEST FOR .NET DEVELOPERS – ICT SYSTEM

Dr. Nick Lehtola

CONTENTS

1. Introduction	3
2. Problem Specification and Requirements	4
2.1. Original Problem.....	4
2.2. Input Representation.....	4
2.3. Test Cases and Expected Results	4
2.4. Naming Convention in the Problem Domain.....	4
3. Design Considerations.....	6
3.1. Considerations and Constraints.....	6
3.2. Classes of Problems	6
3.3. Data Structures.....	7
3.4. Algorithms	7
4. Technical Design	8
4.1. Design Overview	8
4.2. ICT Essentials	8
4.2.1. ICT Core	9
4.2.2. ICT Collections	9
4.2.3. ICT Models.....	9
4.3. ICT Applications	10
4.3.1. How to build and run the ICT.Console.App	10
4.4. Additional Technical Information	10
5. References.....	11

1. Introduction

This document provides information about the strategies, concepts and procedures used in the implementation of the **INTERGEN Coding Test System** (or **ICT System** for short). This system was implemented during the author's interview process with **INTERGEN** (www.intergen.co.nz).

The design and implementation of the **ICT System** was driven by the specifications provided in the **INTERGEN Coding Test for .NET Developers** document. These specifications are explained in details in the **Section 2 (Problem Specification and Requirements)** of this document.

This document is broken down into different sections, briefly described in the table below:

Problem Specification and Requirements	This section describes the original problem and specifications provided in the original INTERGEN Coding Test for .NET Developers document.
Design Considerations	This section describes the considerations, constraints and, as well as, assumptions used in the design and implementation of the ICT System
Technical Design	This section highlights some of the design methodology used in the implementation of the ICT System .
References	Technical references

Table 1. INTERGEN Coding Test Documentation: Main Sections

2. Problem Specification and Requirements

This section describes the original problem and specifications provided in the original **INTERGEN Coding Test for .NET Developers** document.

2.1 Original Problem

The local commuter railroad services a number of towns in NZ. Because of monetary concerns, all of the tracks are 'one-way.' That is, a route from Kaitaia to Invercargill does not imply the existence of a route from Invercargill to Kaitaia. In fact, even if both of these routes do happen to exist, they are distinct and are not necessarily the same distance!

The purpose of this problem is to help the railroad provide its customers with information about the routes. In particular, you will compute the distance along a certain route, the number of different routes between two towns, and the shortest route between two towns. The solution must be flexible enough such that when new routes are added, no code changes are required to be made to the application.

2.2 Input Representation

The input for the problem is considered to be a directed graph where a node represents a town and an edge represents a route between two towns (e.g., AB5, BC4, CD8, DC8, DE6, AD5, CE2, EB3, AE7). The weighting of the edge represents the distance between the two towns. A given route will never appear more than once, and for a given route, the starting and ending town will not be the same town. From the user's standpoint, this graph structure is provided as a single string, in which the interconnections between towns (or cities) are separated by spaces. **Figure 1** shows the default input graph used for testing the correctness of the **ICT System**:

AB5 BC4 CD8 DC8 DE6 AD5 CE2 EB3 AE7

Figure 1. Default Input Graph Used for Testing the ICT System

2.3 Test Cases and Expected Results

The **Table 2** shows some of the testing cases used to verify the correctness of the **ICT System**. The expected results are based on the default input graph discussed in the **Item 2.2**.

2.4 Naming Convention in the Problem Domain

It's important, at this stage, to mention the naming convention used in the *problem domain*. Common definitions used in the problem domain are:

- **City (or town):** It is the primary entity in the problem domain. It defines a location that needs to be serviced by the railroad system. In the current context, a city is represented by a single capital letter (e.g., **A**, **B** or **C**).
- **Link (or city link):** It represents the connection between two cities. As explained in the original problem, a link is always one-directional, meaning that it's not possible to go from city **A** to **B** and come back using the same link (of course, there can be a link that connects **B** to **A**). Links

are represented by two capital letters followed by an integer number (e.g., **AB5**, **DE6**, and **AD5**). The first letter represents the departure city and the second letter represents the destination city. The integer number represents the distance between the cities.

- **Route:** It represents a collection of links and cities that connect two target cities. It's the path that goes from the departure city to the destination city, including all the intermediate ones in between. For instance, using the default input graph, a possible route to go from **A** to **C** is **A-D-C**, using the city-based specification or **AD-DC** using the link-based specification. **Note: A link can be considered as a particular case of a route (a route with only one link).**
- **Network:** It represents the collection of all cities and links in the transit system.

ID	Test Case	Expected Output
1	The distance of the route A-B-C	9
2	The distance of the route A-D	5
3	The distance of the route A-D-C	13
4	The distance of the route A-E-B-C-D	22
5	The distance of the route A-E-D	NO SUCH ROUTE
6	The number of trips starting at C and ending at C with a maximum of 3 stops. In the sample data below, there are two such trips: C-D-C (2 stops) and C-E-B-C (3 stops)	2
7	The number of trips starting at A and ending at C with exactly 4 stops. In the sample data below, there are three such trips: A to C (via B,C,D); A to C (via D,C,D); and A to C (via D,E,B).	3
8	The length of the shortest route (in terms of distance to travel) from A to C.	9
9	The length of the shortest route (in terms of distance to travel) from B to B.	9
10	The number of different routes from C to C with a distance of less than 30. In the sample data, the trips are: CDC, CEBC, CEBCDC, CDCEBC, CDEBC, CEBCEBC, CEBCEBCEBC.	7

Table 2. Original Test Cases and Expected Results

3. Design Considerations

This section describes the considerations, constraints and, as well as, assumptions used in the design and implementation of the **ICT System**.

3.1 Considerations and Constraints

The considerations and constraints addressed in the design and implementation of the **ICT System** are summarized in the **Table 3**. To facilitate the process of identifying the items in the table, each item is associated to an id (e.g., **C1**, **C2**, etc.)

ID	Considerations and Constraints
C1	A city can't be connected by itself through a single link. It can't be self-referenced. Therefore, a link such as AA5 is not valid.
C2	A city can't be isolated. In other words, any city must be part of at least one link in the problem domain, either being the departure city or the destination one (e.g., AB5 or CA7)
C3	The name of the city is represented by a single capital letter (e.g, A , B or C).
C4	The name of a city is unique in the context of the network.
C5	The id of a link is unique in the context of the network.
C6	The distances in the problem domain are represented as integer numbers.
C7	The links are one-directional. For instance, the link AB5 represents the link from A to B. In order to navigate back from B to A , it is necessary to have another link (e.g., BA7) to represent that option.
C8	Based on S5 , the routes are also one-directional.
C9	The network of interconnected cities can be represented by one and only graph structure.
C10	The distance associated to a link is always a positive number.

Table 3. Considerations and Constraints in the ICT System

3.2 Classes of Problems

Based on the specification of the original problem (**Item 2.1**) and the basic test cases presented in the **Table 2**, it's possible to conclude that there are four main classes of problems that need to be addressed in the design and implementation of the **ICT System**. These classes of problems are summarized in the **Table 4**.

The data structure used to represent the network and the algorithms used to address the problems in the **Table 4** are discussed in the next section.

ID	Class of Problem
A	Verify the existence of a given route.
B	Calculate the distance of a given route.
C	Calculate the number of routes respecting imposed constraints (e.g., maximum number of stops or maximum distance travelled).
D	Find the shortest traveled distance between two cities.

Table 4. Classes of Problems Addressed by the ICT System

3.3. Data Structures

In the **ICT System**, the **graph** data structure (Bailey, 2003) is used to represent the problem domain entities (cities, links, routes and network) and their relationships. More specifically, the graph data structure is used to represent the **network** of cities and links. In the context of the *solution domain* (algorithms and implementation) each **vertex** of the graph is associated to a **city**, each **edge** of the **graph** is associated to a **link**, and the **routes** are represented as **paths**.

The graph data structure presents a container to store the vertices and another one for the edges. In order to minimize the access time to these entities **dictionaries** are used as the containers for the vertices and edges. Due to the nature of the specifications and the problems to be addressed, a **directed** and **weighted** graph is used. The weights or costs are stored in the edges. Also, the representation of this graph data structure uses the **adjacency list representation** (Mitchell, 2005). A direct consequence of this approach is that each vertex in the graph contains a list of edges that are incident from this node (outgoing edges).

The graph is the main data structure used throughout the **ICT System**, but it's not the only one. Stacks, dynamic lists and arrays are largely used as well.

3.5. Algorithms

The common thread among the class of problems listed in the **Table 4** is that, in one way or another, their solutions require the use specific techniques to traverse the graph data structure. From the simplest traversal approach (sequential traversal using incident edges from the vertices) to more elaborated ones (such as the DFS - Depth First Search), the algorithmic solutions implemented in the **ICT System** use some form of traversal technique.

The **Dijkstra's algorithm** (Kreyszig, 1988), for example, is implemented for addressing problems of the **Class D (Table 4)**. For the problems of **Class C**, different flavors of the **DFS algorithm** is in place (Tremblay and Cheston, 2003).

For the sake of promoting a higher level of modularity in the **ICT System**, the class that implements the graph data structure (**Graph** class) is not the same as the one that implements the algorithms to manipulate/use the data structure (**GraphAlgorithms** class). In fact, the implementation of the algorithms and their variants is done using extension methods for the **Graph** class.

4. Technical Design

This section highlights some of the design methodology used in the implementation of the **ICT System**.

4.1 Design Overview

The core design of the **ICT System** is based on two distinct architectural segments as shown in Figure 2.

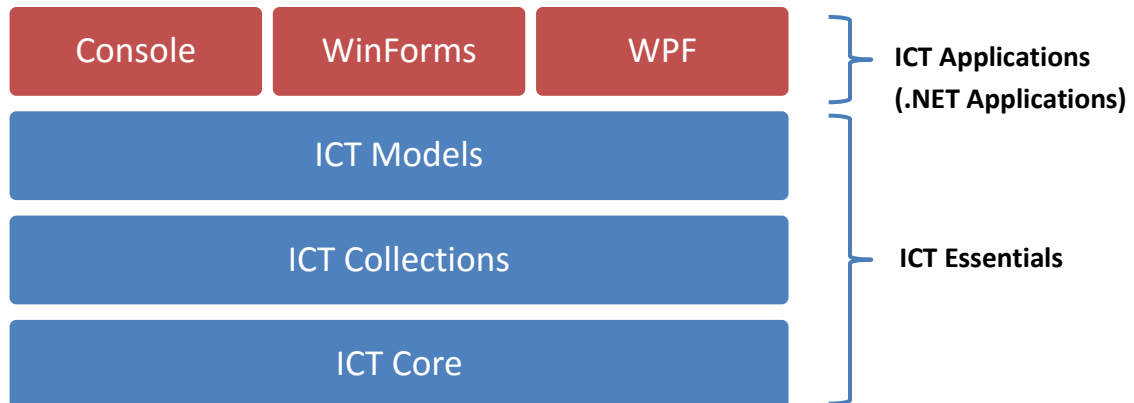


Figure 2. The Fundamental Architectural Segments of the ICT System

The **ICT Applications** is the segment where the **ICT**-based applications are implemented and reside. In theory, these applications are considered to be **.NET** applications and are subjected to the same degree of freedom, portability and constraints as any other **.NET** applications. The implementation of the applications in the **ICT Applications** segment is supported by the **ICT Essentials**. The **ICT Essentials** API provides the backbone support for the development of **ICT**-based applications.

4.2 ICT Essentials

The **ICT Essentials** encompasses all the supporting modules necessary to implement **ICT**-based applications (**.NET**). The **ICT Essential Modules** are listed in the Table 5.

Module	Description
ICT Core	This module contains classes and basic services used by the other modules in the ICT System .
ICT Collections	This module contains classes that implement data structures and algorithms that are used by the other modules of the ICT System .
ICT Models	This module contains classes that implement the basic entities of the problem domain (city, link, route and network). Also, this module provides the API that supports applications that address planning problems (such as the original problem discussed in Section 2 of this document).

Table 5. The ICT Essentials

4.2.1 ICT Core

This module contains classes and basic services used by the other modules in the **ICT System**. Clients could be modules from the **ICT Essentials** segment and/or applications from the **ICT Applications** segment.

There is not just one single common thread of functionalities or services provided by the **ICT Core**. In fact, the services are diversified. Nevertheless, semantically these functionalities and the services are considered to be fundamental to the other modules and applications. For instance, one of the services provided by the **ICT Core** is the lightweight support for **DbC (Design-by-Contract)** (Jazequel and Meyer, 1997). The **Contract** class contains methods for checking *preconditions* (**Requires**) and *post-conditions* (**Ensures**).

4.2.2 ICT Collections

This module contains classes that implement the data structures and algorithms that are used by the other modules of the **ICT System**. Like in the case of the **ICT Core**, the clients of this module could be modules from the **ICT Essentials** segment and/or applications from the **ICT Applications** segment.

The **ICT System** uses the graph data structure to represent the network of cities and links. The **Graph** class implements this data structure. The current implementation is based on the assumptions that the graph is directed and weighted. Because of the adoption of the adjacency list representation, the **Vertex** class has a dynamic container (**List<IEdge<T>>**) to store the incident edges from each **Vertex** instance. Also, based on the design considerations listed in the **Table 3**, the weight of each edge is represented as a positive integer number. The **Edge** class has an integer property that represents the weight (**Weight** property).

In the current design, the algorithms that manipulate and use the graph data structure are not defined in the **Graph** class. A separate class (**GraphAlgorithms**) is used for this purpose. The algorithms are implemented as extension methods in the **GraphAlgorithms** class. This dissociation creates a clear separation in terms of the scope of concerns. One class handles the graph data structure representation (**Graph** class), and the other handles the implementation of the algorithms (**GraphAlgorithms** class).

IMPORTANT: Unit tests are available in the **ICT.Collection.Test** project. These tests were designed to check the behavior of the **ICT Collections** data structures, as well as, the implementation of the graph algorithms. In particular, the test file **GraphAlgorithmsQuestionsTests.cs** checks the behavior and correctness of the **Graph** and **GraphAlgorithms** classes. The tests implemented in the **GraphAlgorithmsQuestionsTests.cs** use the test cases of **Table 2** as references.

4.2.3 ICT Models

This module contains classes that implement the basic entities of the problem domain, and provides the API that supports the implementation of applications at the **ICT Applications** level.

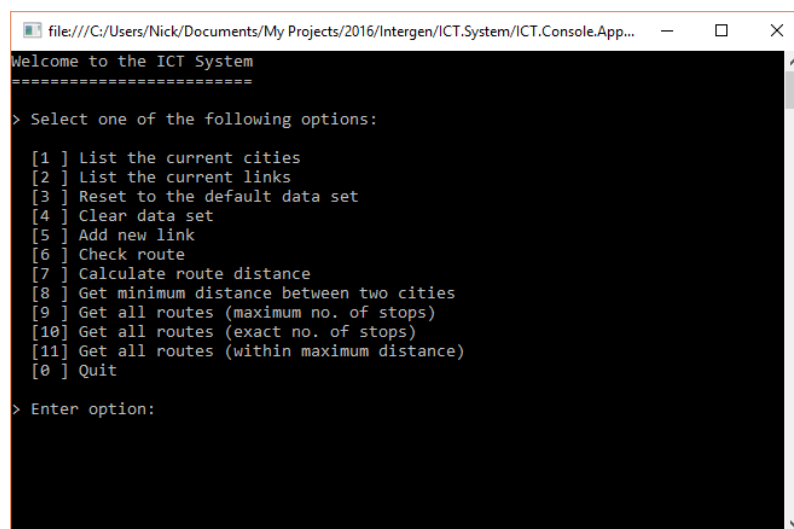
One of the main classes in this module is the **Planner** class. This class serves as the middleman between the applications and the **ICT Essentials**. This class defines the specification of the API used for developing the applications at the **ICT Applications** level. The **Planner** class contains containers

for storing the information about the cities (as instances of the **City** class) and the links (as instances of the **Link** class). On top of that, this class has a property of the type **Graph**. This property (called **Network**) is configured based on the data coming from the city and link containers. Once the **Network** is configured (or in other words, the graph data structure is populated), the **Planner** is able to answer requests from the application client such as the ones listed in **Table 2**.

For more information about the **ICT Essentials API**, check the file **IPlanner.cs**. For more information about the **Planner** class, check the file **Planner.cs**.

4.3 ICT Applications

The **ICT Applications** is the segment where the **ICT**-based applications are implemented and reside. For this particular assessment, a simple console application (**Figure 3**) was implemented to test the core functionalities of the **ICT System**, both in the architectural level, as well as, in the functional/solution level. **Figure 3** shows the main options available in the **ICT.Console.App**.



```
file:///C:/Users/Nick/Documents/My Projects/2016/Interger/ICT.System/ICT.Console.App...
Welcome to the ICT System
=====
> Select one of the following options:

[1 ] List the current cities
[2 ] List the current links
[3 ] Reset to the default data set
[4 ] Clear data set
[5 ] Add new link
[6 ] Check route
[7 ] Calculate route distance
[8 ] Get minimum distance between two cities
[9 ] Get all routes (maximum no. of stops)
[10] Get all routes (exact no. of stops)
[11] Get all routes (within maximum distance)
[0 ] Quit

> Enter option:
```

Figure 3. Options Available in the ICT.Console.App

4.3.1 How to build and run the ICT.Console.App

The following steps should be used to build and run the **ICT.Console.App**:

1. Open the solution file (ICT.sln) in Visual Studio 2012.
2. In the **Solution Explorer** window, right-click on the project named **ICT.Console.App** and select **Set as StartUp Project**
3. Rebuild the solution
4. In main menu, click on the **Start** button (or just press **F5**)

4.4 Additional Technical Information

The **ICT System** was implemented using the **Microsoft Visual Studio 2012**, the **C#** programming language and the **.NET Framework 4.5**.

5. References

Bailey, D. A., *“Java Structures – Data Structures in Java for the Principled Programmer”*, 2nd Ed., McGraw-Hill, 2003.

Jazequel, J.-M. and Meyer, B., *“Design by Contract: the Lessons of Ariane”*, Computer (Volume:30 , Issue: 1), IEEE Computer Society, 1997.

Kreyszig, E., *“Advanced Engineering Mathematics”*, 6th Ed., John Wiley & Sons, 1988.

Mitchell, S., *“An Extensive Examination of Data Structures Using C# 2.0”*, MSDN books, [https://msdn.microsoft.com/en-us/library/aa287104\(v=vs.71\).aspx](https://msdn.microsoft.com/en-us/library/aa287104(v=vs.71).aspx), 2005

Tremblay, J. and Cheston, G. A., *“Data Structures and Software Development in an Object-Oriented Domain”*, Java Edition, Prentice Hall, 2003.