Homework: Data Structures Efficiency

This document defines the **homework assignments** for the "Data Structures" course @ Software University. Please submit a single **zip** / **rar** / **7z** archive holding the solutions (source code) of all below described problems.

Problem 1. Students and Courses

A text file **students.txt** holds information about students and their courses in format **{first-name | last-name | course-name}** like in the example below. Write a program to find and print the **courses** in **alphabetical order** and for each course print its **students ordered by** last name and then by first name.

Input	Output
Stela | Mineva | Java Yasen | Ivanov | C# Stefka | Nikolova | SQL Miroslav | Tsenov | SQL Milena | Petrova | C# Asya | Yankova | SQL Ivan | Grigorov | C# Ivan | Kolev | SQL Vanya | Angelova | Java Todor | Georgiev | SQL	C#: Ivan Grigorov, Yasen Ivanov, Milena Petrova Java: Vanya Angelova, Stela Mineva SQL: Todor Georgiev, Ivan Kolev, Stefka Nikolova, Miroslav Tsenov, Asya Yankova

Hints:

- Define a class **Person {FirstName, LastName}** to hold persons.
- Implement **IComparable<Person>** in the **Person** class to enable ordering of persons by their last name.
- Use **SortedDictionary<string, SortedSet<Person>>** to map courses to persons.
- **Read** and **parse the input** and put each input line format **{first-name | last-name | course-name}** into the dictionary:
  - When the **course-name** does not exists as key in the dictionary, add a new **SortedSet<Person>** for this **course-name** to the dictionary.
  - Add the **Person {first-name, last-name}** to the dictionary by key **course-name**.
- **Enumerate** and **print** all dictionary items. Each dictionary item holds the **course name** as key and a **set of persons** as value.

Problem 2. Implement BiDictionary<K1, K2, T>

Implement a class **BiDictionary<K1, K2, T>** that allows adding **triples {key1, key2, value}** and **fast search** by **key1**, **key2** or by both **key1** and **key2**. Allow multiple values to be stored for given key.

You may **use multiple hash tables** behind and finish the following code:

```
public class BiDictionary<K1, K2, T>
{
    private Dictionary<K1, List<T>> valuesByFirstKey;
    private Dictionary<K2, List<T>> valuesBySecondKey;
    private Dictionary<Tuple<K1, K2>, List<T>> valuesByBothKeys;

    public void Add(K1 key1, K2 key2, T value) { … }
    public IEnumerable<T> Find(K1 key1, K2 key2) { … }
    public IEnumerable<T> FindByKey1(K1 key1) { … }
```

```
    public IEnumerable<T> FindByKey2(K2 key2) { … }
    public bool Remove(K1 key1, K2 key2) { … }
}
```

Example of using the **BiDictionary<K1, K2, T>**:

```
var distances = new BiDictionary<string, string, int>();
distances.Add("Sofia", "Varna", 443);
distances.Add("Sofia", "Varna", 468);
distances.Add("Sofia", "Varna", 490);
distances.Add("Sofia", "Plovdiv", 145);
distances.Add("Sofia", "Bourgas", 383);
distances.Add("Plovdiv", "Bourgas", 253);
distances.Add("Plovdiv", "Bourgas", 292);
var distancesFromSofia = distances.FindByKey1("Sofia"); // [443, 468, 490, 145, 383]
var distancesToBourgas = distances.FindByKey2("Bourgas"); // [383, 253, 292]
var distancesPlovdivBourgas = distances.Find("Plovdiv", "Bourgas"); // [253, 292]
var distancesRousseVarna = distances.Find("Rousse", "Varna"); // []
var distancesSofiaVarna = distances.Find("Sofia", "Varna"); // [443, 468, 490]
distances.Remove("Sofia", "Varna"); // true
var distancesFromSofiaAgain = distances.FindByKey1("Sofia"); // [145, 383]
var distancesToVarna = distances.FindByKey2("Varna"); // []
var distancesSofiaVarnaAgain = distances.Find("Sofia", "Varna"); // []
```

Problem 3. Collection of Products

A large trade company has millions of **products**, each described by **id** (unique), **title**, **supplier** and **price**. Implement a data structure to store them that allows:

- **Add** new product (if the **id** already exists, the new product replaces the old one)
- **Remove** product by **id** – returns **true** or **false**
- **Find products** in given **price range** [x…y] – returns the products sorted by **id**
- **Find products** by **title** – returns the products sorted by **id**
- **Find products** by **title + price** – returns the products sorted by **id**
- **Find products** by **title + price range** – returns the products sorted by **id**
- **Find products** by **supplier + price** – returns the products sorted by **id**
- **Find products** by **supplier + price range** – returns the products sorted by **id**

Hints:

- **Combine multiple data structures** for best performance of the individual operations.
- Define class **Product** to hold product **id** + **title** + **supplier** + **price**.
- You may use **Dictionary<string, SortedSet<Product>>** to map products by various keys (e.g. title + price).
- You may use **OrderedMultiDictionary<price, Product>** from Wintellect's Power Collections for .NET to map product price to products.