Implement a B-Tree that stores telephone contacts on disk, with the restriction that only a single file can be used. Keeping in mind that each contact has a name, a telephone number and their age, you should be able to add contacts and respond to the following requests:

Find all contact that:

1) Given a string s, s be prefix of its name.

2) Given a string s, s be the prefix of your phone.

3) Given a positive integer n, your age is n.

4) Given a positive integer n, its age is less strict than n.

5) Given a positive integer n, your age is more strict than n.

Implementation:

You must create a .NET library (.dll) that contains a class that implements the IContactsStore interface. The implementation of your B-Tree should also be in that library. For this use the template provided and do not add another project to the solution.

```
Public interface IContactsStore
{
    void Open(Stream db);
    void Close();
    void AddContact(IPerson person);
    IEnumerable<IPerson> FindByName(string preffixName);
    IEnumerable<IPerson> FindByPhone(string preffixPhone);
    IEnumerable<IPerson> FindByAge(int age);
    IEnumerable<IPerson> FindYounger(int age);
    IEnumerable<IPerson> FindOlder(int age);
}
```

```
public interface IPerson
{
    string Name { get; set; }
    string Phone { get; set; }
    int Age { get; set; }
}
```

Considerations:

1) The name of a person (Name) will only contain the characters: {'0' - '9', 'A' - 'Z', 'a' - 'z', ''} and will not exceed 50 bytes (ASCII characters).

2) A person's phone number will only contain the characters: {'0' - '9'} and will never start with '0'.

3) The age (Age) of a person will only contain the characters: {'0' - '9'} and will never start with '0'.

4) The FindByName method returns the contacts sorted by name. The FindByPhone method returns contacts ordered by phone. The FindByAge, FindYounger and FindOlder methods return contacts sorted by age. If the criterion by which two contacts are being compared is the same, then the one that was first inserted is first ordered.

5) You should not consider the names, telephone numbers and age of the contacts, these values may not represent data that correspond to reality (EXAMPLE: A person's age can be '999', their phone can be '666 'and his name' Hi123 ').

Specifications:

1) You should be able to control the memory used by your program.

2) Any method that returns an IEnumerable must do it in a lazy way, that is, you should not load all the contacts of an order in memory and then return them.

3) The Stream that will be provided through the Open method is used to not keep all contacts in memory.

4) Only basic data methods and structures of the .NET platform can be used (variables, arrays, mathematical operations, etc.). Sorting methods and dictionary type data structures are examples of elements that can not be used.

Examples:

1) The EDA_PROJECT_1314_demo solution contains two projects: the first one is BruteContactsStore that makes a brute force implementation of the IContactsStore and Visualizer interface, a Windows Form project.

When you run Visualizer, an instance of BruteContactsStore is created and the contacts that are in the file EDA_PROJECT_1314_demo \ Visualizer \ resources \ db_test.txt are inserted. Note that this example does not use Stream, so the contacts must be inserted every time the Visualizer project is executed.