$$S(n) = n!\cdot\left(1+\frac{1}{n}+O\left(\frac{1}{n^2}\right)\right) \quad (n \to \infty).$$

Литература

1. Панченков, А.Н. Асимптотические методы в экстремальных задачах механики. – Новосибирск: Наука, 1982.

2. Олвер, Ф. Введение в асимптотические методы и специальные функции. – М.: Наука, 1978.

3. Грэхем, Р. Конкретная математика. Основание информатики: Пер. с англ. / Р. Грэхем, Д. Кнут, О. Паташник. – М.: Мир, 1998.

Д.Е. Яблоков

СПЕЦИАЛИЗАЦИЯ ПОВЕДЕНИЯ ИНТЕРВАЛЬНЫХ АЛГОРИТМОВ С ПОМОЩЬЮ МОДЕЛЕЙ ОБОБЩЕННЫХ КОНЦЕПЦИЙ ИТЕРАТОРОВ

(Самарский национальный исследовательский университет
имени академика С.П. Королева)

Назначение абстрактного типа данных состоит в расширении предлагаемого языком программирования понятийного аппарата в соответствии с контекстом предметной области. Создание какого-либо интерфейса подразумевает, что как базовая абстракция он должен помогать формировать правильные стереотипы мышления и, соответственно, правильный стиль программирования. По сути, проектирование интерфейсов – это проектирование конструкций предметно-ориентированного языка, представленного с помощью синтаксических средств используемого инструмента кодирования. Таким образом, обеспечивается взаимосвязь используемых абстракций и требований, как предметной области, так и выбранной стратегии реализации. Но способ выражения соответствия требованиям при разных подходах имеет существенные отличия. В объектно-ориентированном программировании для любого типа, находящегося в каком-либо дочернем узле иерархии наследования, необходимо реализовать определенный его интерфейсом контракт. В терминах этого контракта, следуя логике объектно-ориентированного подхода, возможно обращение к функциональности и данным экземпляра потомка. Обобщенное программирование подразумевает, что проектируемый элемент программы удовлетворяет необходимому набору свойств и ограничений, которые предъявляются к нему со стороны других компонентов, ожидающих определенный формат взаимодействия сообразно своим синтаксическим и семантическим особенностям.

Алгоритм, написанный в обобщенном стиле, может применяться для любых типов, удовлетворяющих требованиям, которые он предъявляет к своим аргументам. Любой обобщенный алгоритм состоит из двух частей: конкретных инструкций, определяющих шаги исполнения и совокупности концепций, по-

средством которых происходит спецификация абстрактных требований. Например, алгоритм Search, в представленном фрагменте кода (Рис. 1), ищет подпоследовательность первого диапазона [`first1`; `last1`), которая идентична второму диапазону [`first2`; `last2`) при их поэлементном сравнении. Требования, которые предъявляются к его аргументам, задающим границы диапазонов, должны быть сформулированы таким образом, чтобы отражать все необходимые функциональные особенности. Модели однонаправленных, неизменяющих итераторов `IFI1`, `IFI2` должны быть способны:

1. создавать собственную копию с сохранением состояния, т.е. текущей позиции (`first1.Clone()`);
2. поддерживать отношение эквивалентности, т.е. быть сравнимыми на равенство позиций (`first2.EqualTo(last2)`) относительно начала диапазона;
3. выполнять операцию продвижения (`interator11.Advance()`) в прямом направлении с единичным шагом, после которой итератор должен указывать на следующую позицию диапазона относительно предыдущего состояния;
4. прочесть значение элемента (`iterator2.Read()`), находясь в какой-либо позиции диапазона не совпадающей с его окончанием (`last2`).

```
public static IFI1 Search<IFI1, IFI2, T1, T2>
(
    IFI1 first1,
    IFI1 last1,
    IFI2 first2,
    IFI2 last2,
    IBinaryPredicate<T1, T2> predicate
)
    where IFI1 :
IInputForwardIterator<T1>
    where IFI2 :
IInputForwardIterator<T2>
{
    IFI1 iterator1 = (IFI1)first1.Clone();
    if(first1.EqualTo(last1) || first2.EqualTo(last2))
        return iterator1;
    for(; ; iterator1.Advance())
    {
        IFI1 iterator11 = (IFI1)iterator1.Clone();
        for(IFI2 iterator2 = (IFI2)first2.Clone(); ;
            iterator11.Advance(), iterator2.Advance())
        {
            if(iterator2.EqualTo(last2))
                return iterator1;
            else if(iterator11.EqualTo(last1))
                return last1;
            else if(!predicate.Execute(iterator11.Read(),
                                       iterator2.Read()))
                break;
        }
    }
}
```

Рис. 1. Алгоритм поиска поддиапазона с использованием моделей однонаправленных, неизменяющих итераторов на языке C#

Все эти требования к поведению однонаправленных, предоставляющих доступ к элементам только в режиме чтения итераторов, задаются с помощью ограничений для типов данных, определенных в универсальном объявлении параметров алгоритма (`where IFI1 : IInputForwardIterator<T1>` и `where IFI2 : IInputForwardIterator<T2>`). Формализация требования с помощью механизма ограничений для универсальных типов данных выбрана потому, что в качестве параметра типа модели итератора может быть передана абстракция с более развитой семантикой, которая в свою очередь, может использоваться далее в вызывающем коде.

```
var iRange1 = ContainerFactory.IStringRange("Hello, world!");
var iRange2 = ContainerFactory.IStringRange("world");
var result = AlgorithmFactory.Search
                                (
                                   iRange1.Begin,
                                   iRange1.End,
                                   iRange2.Begin,
                                   iRange2.End,
                                   OperationFactory
                                      .ToPredicate<char>((lhs, rhs)
                                              => (lhs == rhs))
                                );
Assert.AreEqual(result.Distance(iRange1.Begin), 7);
```

Рис. 2. Пример применения алгоритма поиска поддиапазона с использованием итераторов произвольного доступа

Как видно из примера (Рис. 2), при проведении unit-теста для верификации полученного значения необходимо вычислить расстояние между итератором произвольного доступа, полученным в результате работы алгоритма (`result`) и итератором, обозначающим начало диапазона (`result.Distance(iRange1.Begin)`). Сделать это было бы невозможно, если бы тип возвращаемого значения и сигнатура алгоритма были определены в терминах базовых абстракций однонаправленного, неизменяющего итератора (`IInputForwardIterator<...>`). Ограничения, накладываемые на типы аргументов алгоритма поиска подпоследовательности (Рис. 1), которому для работы достаточно функциональности итератора прямого доступа, никак не должны влиять на тип моделирующий концепцию такого итератора. Отношения между концепциями должны формализоваться процессом развития одной концепции от другой, когда развивающая концепция предоставляет всю функциональность развиваемой и, возможно, некоторую дополнительную, являющуюся только частью развивающей. Практически это означает, что если алгоритм требует, чтобы его формальные параметры являлись моделью какой-либо концепции, то его фактические параметры могут представлять собой тип, являющийся моделью другой концепции, при условии, что она развивает концепцию, определяющую набор требований к формальным параметрам алгоритма.

Модель итератора произвольного доступа максимально приближена к семантике арифметики указателей и поддерживает наибольшее подмножество

операций, часть из которых уже определена в развиваемых ею концепциях итераторов. При реализации алгоритма, требующего от своих аргументов семантики произвольного доступа, нужно определить ограничение для параметров универсальных типов с помощью обобщенной концепции неизменяющего итератора произвольного доступа (`where IRI1 : IInputRandomAccessIterator<D, T1>`) и `where IRI2 : IInputRandomAccessIterator<D, T2>`). Здесь `D` – обобщенный тип для представления числа элементов между итераторами, указывающими на разные позиции одного диапазона. Спецификация поведения данного типа, т.е. перечень доступных для него операций передается в алгоритм с помощью экземпляра обобщенной концепции `IBinaryDistance<D, D, D>`.

```
public static IRI1 Search<IRI1, IRI2, D, T1, T2>
(
    IRI1 first1,
    IRI1 last1,
    IRI2 first2,
    IRI2 last2,
    IBinaryDistance<D, D, D> concept,
    IBinaryOperation<T1, T2, bool> predicate
)
    where IRI1 :
IInputRandomAccessIterator<D, T1>
    where IRI2 :
IInputRandomAccessIterator<D, T2>
{
    D count1 = last1.Distance(first1);
    D count2 = last2.Distance(first2);
    IRI1 iterator1 = (IRI1)first1.Clone();
    for(; concept.LessEqual(count2, count1);
        iterator1.Advance(),
            count1 = concept.Minus(count1,
                                   concept.One))
    {
        IRI1 iterator11 = (IRI1)iterator1.Clone();
        for(IRI2 iterator2 = (IRI2)first2.Clone(); ;
                iterator11.Advance(), iterator2.Advance())
        {
            if(iterator2.EqualTo(last2))
                return iterator1;
            else if(!predicate.Execute(iterator11.Read(),
                                       iterator2.Read()))
                break;
        }
    }
    return last1;
}
```

Рис. 3. Алгоритм поиска поддиапазона с использованием моделей неизменяющих итераторов произвольного доступа на языке C#

Использование обобщенных концепций итераторов предполагает реализацию обобщенных интервальных алгоритмов (Рис. 1, 3). При работе с любыми структурами данных создание таких алгоритмов требует меньше усилий, а код выглядит более понятным и логичным.

Литература

1. Страуструп, Б. Программирование. Принципы и практика с использованием C++. Второе издание / Б. Страуструп: Пер. с англ. – М.: Издательство Вильямс – 2016. – 1328 с.

2. Шилдт, Г. C# 4.0 Полное руководство. Отдельное издание / Г. Шилдт: Пер. с англ. – М.: Издательство Вильямс – 2015. – 1056 с.

3. Яблоков, Д.Е. Применение обобщенных концепций итераторов и функциональных адаптеров для создания алгоритмов с развитой семантикой обработки данных. МНТК «Перспективные информационные технологии»: Сб. науч. тр. / под ред. С.А. Прохорова. Самара: СГАУ – 2017. – С. 1041 -1044.