ЛЕКЦИЯ 1

- Функции на последовательностях
- Индуктивные функции, алгоритм вычисления индуктивных функций
- Индуктивные рашириения не индуктивных функций

Понятие функции на последовательностях

Пусть задана последовательность объектов некторого типа:

$$a_1, a_2, ..., a_n,\ где\ a_k \in \Omega - некоторое\ множество\ (определяющее\ тип\ последовательности)$$

Будем использовать обозначения:

$$A=[a_1,a_2,...,a_n]\in \Omega^n=\underbrace{\Omega\times...\times\Omega}_n$$

$$\Omega^0=\varnothing,\ [\ ] - пустая\ последовательность,\ [\ ]\in \overline{\Omega}$$

$$\overline{\Omega}=\bigcup_{n=0}^{\infty}{\Omega^n}$$

Тогда $$F:\overline{\Omega}\rightarrow U,\ где\ U - некоторое\ множество$$

-- это есть некоторая функция, определенная на всех последовательностях конечной длины данного типа.

Примеры функций на последовательностях и алгоритмов их вычисления

1. Длина последовательности: $$ F(A)=length(A) \in \N \cup {0} $$ Вычисление этой функции сводится к вычислению `n`-го члена следующей вспомогательной последовательности: $$ l_0=length([\ ])=0\ ....\ l_k=l_{k-1}+1\ ....\ l_{n}=l_{n-1}+1 $$ Соответствующий программный код на языке Julia выглядит так:

```
function length(A)
    len = 0
    for _ in A
        len += 1
    end
    return len
end
```

2. Сумма членов последовательности: $$ F(A)=sum(A)=sum_{k=1}^na_k\ \in U=\Omega. $$ Вычисление этой функции сводится к вычислению n-го члена следующей вспомогательной последовательности: $$ s_0=sum([\ ])=0\ ....\ s_{k}=s_{k-1}+a_k\ ....\ s_{n}=s_{n-1}+a_n $$

Соответствующий программный код на языке Julia выглядит так:

```
function sum(A)
    s = eltype(A)(0)
    for a in A
        s += a
    end
    return s
end
```

3. Произведение членов последовательности: $$ F(A)=prod(A)=a_1 \cdot\ ... \cdot a_n=\prod_{k=1}^na_k\ \in U=\Omega. $$ Вычисление этой функции сводится к вычислению n-го члена следующей вспомогательной последовательности: $$ p_0=prod([\ ])=1\ ....\ p_{k}=p_{k-1} \cdot a_k\ ....\ p_{n}=p_{n-1} \cdot a_n $$

Соответствующий программный код на языке Julia выглядит так:

```
function prod(A)
    p = eltype(A)(1)
    for a in A
        p *= a
    end
    return p
end
```

4. Максимальное значение членов последовательности: $$ F(A)=maximun(A)=max(a_1,...,a_n)\ \in U=\Omega. $$ Вычисление этой функции сводится к вычислению n-го члена следующей вспомогательной последовательности: $$ M_0=maximum([\ ])=-\infty\ ....\ M_{k}=max(M_{k-1},a_k)\ ....\ M_{n}=max(M_{n-1},a_n) $$

Соответствующий программный код на языке Julia выглядит так:

```
function maximum(A)
    M = typemin(eltype(A)) # m = -Inf
    for a in A
        M = max(M,a)
    end
    return M
end
```

5. Минимальное значение членов последовательности: $$ F(A)=minimun(A)=min(a_1,...,a_n)\ \in U=\Omega. $$ Вычисление этой функции сводится к вычислению `n`-го члена следующей вспомогательной последовательности: $$ m_0=minimum([\ ])=\infty\ ....\ m_{k-1}=min(m_{k-1},a_k)\ ....\ m_{n}=min(m_{n-1},a_n) $$

Соответствующий программный код на языке Julia выглядит так:

```
function maximum(A)
    m = typemin(eltype(A)) # m = -Inf
    for a in A
        m = min(m,a)
    end
    return m
end
```

6. Индекс максимального значения членов последовательности: $$ F(A)=argmax(A) $$

- это значение индекса элемента, при котором элемент достигает наибольшего значения, т.е. $$ maximum(A)=A[argmax(A)] $$ Вычисление этой функции сводится к вычислению следующей вспомогательной последовательности: $$ i_{max\ 1}=maximum([a_1])=1\ ....\ i_{max\ k}=A[k]>A[k-1]\ ?\ k : i_{max\ k-1}\ ....\ i_{max\ n}=A[n]>A[n-1]\ ?\ n : i_{max\ n-1} $$

Соответствующий программный код на языке Julia выглядит так:

```
function argmax(A)
    @assert !isempty(A)
    imax = 1
    for k in eachindex(A)
        if A[k] > A[imax]
            imax = k
        end
    end
    return imax
end
```

Аналогично определяется функция argmin(A).

7. Значение многочлена в точке, вычисленное по последовательности его коэффициентов, заданной по убыванию степеней, по хеме Горнера.

$$ P_n(x)=a_0 \cdot x^n + a_1 \cdot x^{n-1} + ... + a_{n-1} \cdot x + a_n $$

Будем считать, что задан массив коэффициентов $A=[a_0,a_1,...a_n]$, и некторое значение аргумента x. Тогда $F(A)=P_n(x).$

Для вычисления этой функции воспользуемся следующей вспомогательной последовательностью многочленов: $$ Q_0(x)=a_0\ Q_1(x)=Q_0 \cdot x + a_1 = a_0 \cdot x + a_1\ ....\ Q_k(x)=Q_{k-1} \cdot x + a_k\ ....\ Q_n(x) = Q_{n-1} \cdot x + a_n = P_n(x) $$ Соответствующий программный код на языке Julia выглядит так:

```
function polyval(A,x)
    Q = a[1] # - это есть a_0
    for a in @view A[2:end]
        Q=Q*x+a
    end
```

```
    return Q
end
```

ЗАМЕЧАНИЕ. Для работы с многочленами в языке Julia имеется специальный пакет (см. https://github.com/JuliaMath/Polynomials.jl)

8. Сортировка массива A, в этом случае F(A)=sort(A) $\in \overline{\Omega}$

Идея сортировки вставками состоит в том, что если первые `k` элементов массива `A` уже отсортированы, то для того, чтобы получить отсортированными первые `k+1` элементов этого массива, достаточно просто вставить уго `k+1`-ый элемент в соответствующую позицию, сдвигая поочередно все элементы, которые больше его, на 1 позицию вправо. Поскольку для k=1 необходимое предположение об отсортированности начальной части массива всегда выполнено, то остается только проитерировать `k` от `2` до `n` (`n=length(A)`.)

Соответствующая функция на языке Julia могла бы выглядеть так:

```
function insertsort!(A)
    n=length(A)
    for k in 2:n
        # часть массива A[1:k-1] уже отсортирована
        op_insert!(A,k)
    end
    return A
end

op_insert!(A,k) =
    while k>1 && A[k-1] > A[k]
        A[k-1], A[k] = A[k], A[k-1]
        k -= 1
    end
```

Индуктивные функции на последовательностях

Зададимся вопросом: что общего у всех рассмотреннных выше функций на последовательностях? Чтобы ответить на этот вопрос с математической точностью, понадобится следующее определение.

Определение. Функция $F:\overline{\Omega}\rightarrow U$ называется **индуктивной** (по А.Г.Кушниренко), если существует такая функция двух переменных (операция) $op: U \times \Omega \rightarrow U$, такая, что для любой последовательности A $\in \overline{\Omega}$ и для любого нового элемента a $\in \Omega$, $$ F([A...,a])=op(F(A),a), $$ где [A...,a]=[A[1],...,A[end],a]

Как можно убедиться все рассмотренные выше функции являются индуктивными.

Так, в случае `F(A)=length(A), op(L,a)=L+1` (зависимость от второго аргумента здесь только формальная);

в случае `F(A)=sum(A), op(s, a) = s+a`;

в случае `F(A)=prod(A), op(p, a) = p*a`;

в случае `F(A)=maximum(A), op(M, a) = max(M, a)`;

в случае `F(A)=minimum(A), op(m, a) = min(m, a)`;

в случае `F(A)=argmax(A), op(imax, A[k]) = A[k] > A[imax] ? k : imax`;

в случае `F(A)=`$P_n(x)$, `op(Q, a) = Q*x + a`;

наконец, в случае `F(A) = insertsort!(A), op(A[1:k], A[k+1]) = end_insert!(A[1:k+1])`.

Можно заметить, что вычисление любой индуктивной функции можно свести к следующей универсальной схеме (алгоритму), точнее говоря, рассматриваемая ниже схема может иметь незначительные вариации, связанные с инициализацией переменной, в которой затем формируется результат вычислений:

```
y = F([ ]) # значение индуктивной функции на пустой последовательности
for a in A
    y = op(y,a)
end
```

Здесь под A понимается некоторый итерируемый объект, представляющий последовательность. Не обязательно это именно массив, это также может быть и кортеж, и диапазон, и генератор, и какой-либо контейнер, например, множество.

При этом функцию `F(A)` необходимо доопределять на пустой последовательности таким значением, нейтральным по отношению к опрерации `op`. Например, при вычислении суммы чисел, таким нейтральным значением будет число `0`, при вычислении произведения чисел, нейтральным значением будет число `1`, при поиске минимального элемента, нейтральным значением будет символ бесконечности, при сортировке массива (вставками) - пустой массив.

Но иногда удобнее обходиться без доопределения вычисляемой функции на пустой последовательности, тогда алгоритм примет вид:

```
y = F([A[1]]) # значение индуктивной функции на подпоследовательности, содержащей только 1-й элемент A
for a in A[2:end]
    y = op(y,a)
end
```

Вообще, начинать вычисления можно не только с пустой последовательности, или с последовательности из одного элемента, но и - с последовательности, содержащей любое другое число начальных элементов.

При программировании вычислений индуктивной функции требуется рассматривать входные данные как последовательность некоторых значений, в требуемом порядке поступающих для "обработки".

ЗАМЕЧАНИЕ. Если функция на последовательности является индуктивной, то в нашем распоряжении имеется рассмотренный здесь универсальный алгоритм ее вычисления. Для записи этого алгоритма требуется только инициализачия начального значения переменной `y` (в нашей записи) и конкретизация операции `op`, т.е. определение соответствующей функции 2-х переменных.

Для программирования вычислений индуктивных функций в языке Julia есть специальная функция высшего порядка(и в Python подобная функция тоже есть):

```
reduce(op, itr; [init])
```

где

- `op` - это аргумент типа Function, определяющий рекурсивную опрерацию, т.е. соответствующую функцию 2-х аргументов;
- `itr` - это итерируемый объект, задающий последовательность (например, одномерный массив);
- `init` - необязательный именованный аргумент, определяющий значение, которым доопределяется вычисляемая индуктивная функция на пустой последовательности (при фактическом отсутствии этого аргумента, вычисление будет осуществляться в соответствии со 2-м вариантом алгоритма).

Например, функция sum, могла бы быть определена следующим образом:

```
sum(A)=reduce(+, A; init=eltype(A)(0))
```

В этом случае `sum([ ])` вернет 0.0 (тип результата будет именно Float64, потому что аргумент eltype([]) имеет значение Float64).

А если определить функция `sum` так:

```
sum(A)=reduce(+,A)
```

то вызов `sum([ ])` приведет к ошибке (функция не определена на пустой последовательности).

Точно также, в одну сточку, можно было бы реализовать и рассмотренную ранее функцию polyval(A,x), вычисляющую значение многочлена в точке по схеме Горнера:

```
polyval(A,x)=reduce((Q,a)->Q*x+a, A)
```

Вычислительная сложность алгоритма

Вычислительную сложность оценивают числом элементарных операций, необходимых для выполнения алгоритма. Под элементарными опрациями обычно понимают 4 арифметических действия `+`, `-`, `*`, `/`, операции сранения `<`, `>`, операцию копирования и т.п.

Необходимый для выполнения алгоритма объём памяти также относят к вычислительной сложности (ресурсоёмкости) алгоритма, выражается в байтах.

Обычно интерес представляет зависимость вычислительной сложности алгоритма от "размера" задачи - некоторого параметра, от которого зависит сложность. Применительно к массивам - это размер (длина) массива, который обычно обозначают буквой `n`.

Зависимость сложности алгоритма от параметра `n` выражают какой-либо функцией от этого параметра.

Обычно получить точную оценку сложности алгоритма затруднительно, или в этом нет необходимости. Как правило получают оценку сложности для "наихудшего случая", т.е. при наименее благоприятных исходных данных, а иногда - делают оценку в среднем по всем возможным данным.

В связи с этим часто при получении оценки сложности алгоритма в виде функциональной зависимости от параметра n интересуются только, как ведет себя эта зависимость при больших n ($n \rightarrow \infty$)

Определение. Говорят, что неотрицательная функция $f(n)=O(g(n)), n \rightarrow \infty$, где $g(n)$ - некотрая другая неотрицательная функция, если существует такая положительная константа $C>0$, и такое натуральное число n_0, что для всех $n>n_0$ $f(n) <= C \cdot g(n)$.

Тогда, например, тот факт, что функция $f(n)$ является ограниченной с помощью символики O-большое можно записать так: $f(n)=O(1)$.

Таким образом, то что алгоритмическая сложность некоторого алгоритма оценивается для наихудшего случая как $O(g(n))$ означает, что число необходимых элементарных операций не превзойдет $C \cdot g(n)$.

Возвращаясь к универсальному алгоритму вычисления индуктивной функции, его вычислительную сложность можно оценить (с верху), как $O(n \cdot g(n))$, где $g(n)$ - оценка вычислительной сложности для вычисления функции op (опрерации). Если сложность этой операции может быть оценена как $O(1)$ (сложность ограничена), то оценка сложности вычисления такой индуктивной функции будет $O(n)$.

Такую (линейную) сложность имеют все рассмотренные выше примеры, за исключением послежднего, т.е. за исключением алгоритма сортировки. Там сложность только одной операции вставки имеет оценку $O(n)$, поэтому оценка сложности всего алгоритма сортировки вставками получается только $O(n^2)$.

Однопроходные алгоритмы

Под однопроходым алгоритмом понимаются алгорим, который выдаёт ответ в результате однократного перебора элементов некоторой последовательности, представляющей исходные данные.

В этом смысле универсальная схема вычисления индуктивной функции на последовательности всегда даёт однопроходный алгоритм, при условии, конечно, что выполняемая на каждом шаге такого алгоритма операция имеет асимптотическую оценку сложности $O(g(n))$ существенно более низкую, чем $O(n)$. Говоря точнее, последнее означает, что $\lim_{n \rightarrow \infty} \frac{g(n)}{n}=0$, например, - $O(1)$. Таким образом, алгоритмы вычисления всех рассмотренных здесь нами индуктивных функций, за исключением сортировки, являются однопроходными.

Сложность операции вставки в алгоритме сортировки вставками оценивается только как $O(n)$, поэтому данный алгоритм не может считаться однопроходным.

При этом, в случае сортировки, существуют и более эффекктивные алгоритмы (в ассимптическом смысле), котрые не будут сводиться к универсальному алгоритму вычисления индуктивной функции, и которые имею асимптотическую оценку сложности $O(n \cdot log(n))$.

Индуктивные расширения неиндуктивных функций

Пусть $$ F(A)=mean(A)=sum(A)/length(A). $$ Эта функция не является индуктивной. Доказать это можно, расуждая от противного. В самом деле, предположим, согласно определению индуктивной функции, что существует такая функция двух переменных $op(m, a)$, такая, что для любых $A \in \overline{\Omega}$ и $a \in \Omega$, $$ mean([A...,a])=op(mean(A),a). $$ Из этого следует, что величины $mean(A)$ и a однозначно определяют результат. Но, с другой стороны, $$ mean([A...,a])=\frac{sum(A)+a}{length(A)+1}=\frac{mean(A)+\frac{a}{length(A)}}{1+\frac{1}{length(A)}}, $$ откуда видно, что значения $mean(A)$ и a сами по себе не определяют величину $mean([A...,a])$: тут имеется еще зависимость от $length(A)$, притом между $mean(A)$ и $length(A)$, очевидно, нет однозначного соответствия. Таким образом, имеем противоречие.

Однако, функции $mean(A)$ и $length(A)$, как мы знаем, сами ао себе являются индуктивными. Поэтому, функция на последовательностях, значениями которой будет пара значений (кортеж значений): $$ F^*(A)=(sum(A), length(A)), $$ уже будет индуктивной. Эту индуктивную функцию можно вычислить с использование универсального алгоритма, а интересующая нас не индуктивная функция $mean(A)$ уже просто выражается через ее компоненты.

Определение. Функция

$$ F*: \overline{\Omega} \rightarrow U^* $$ называется индуктивным расширение неиндуктивной функции $F(A)$, если существует такая функуия $$ P: U^* \rightarrow U $$ такая что для любой последовательности $A \in \overline{\Omega}$ $$ F(A)=P(F^*(A)) $$ *В частности, в случае не индуктивной функции $mean(A)$ её индуктивным расширением будет функция* $$F^*(A)=(sum(A),length(A)). При этом $$ mean(A)=\frac{sum(A)}{length(A)}=\frac{F^*(A)[1]}{F^*(A)[2]}. $$

Простые примеры построения индуктивных расширений неиндуктивный функций

Один такой пример, а именно, функция `mean(A)`, вычисляющая среднее арифметической последовательности уже нами был рассмотрен. Вот еще несколько простых примеров.

Наибольшее значение в заданной последовательности может быть не единственным (может существовать несколько членов последовательности, иеющих максимальное значение).

Пусть требуется посчитать число **максимальных элементов** в заданной последовательности. Число максимальных элементов, очевидно, не является индуктивной функцией на последовательностях. В самом деле, если известно число максимальных элементов в некоторой начальной части заданной последовательности, и получено значение еще одного, следующего, члена последовательности, то не известно, как правильно должно изменится общее число максимумов - это зависит от значения самого максимума: либо новый член последовательности меньше максимума, тогда число максимумов останется прежним, либо он равен прежнему максимуму, и тогда общее число максимумов надо увеличить на 1, либо его значение больше прежнего максимума, и тогда число максимумов должно быть положено равным 1.

Таким образом, ясно, что индуктивным расширением, в данном случае, будет кортеж из двух значений: число максимумов и само максимальное значеие.