МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. Р.Е. АЛЕКСЕЕВА»
Институт радиоэлектроники и информационных технологий
Кафедра «Информационные радиосистемы»

Индуктивные операции и функции высших порядков
(Вариант 28)

Выполнил:
Студент гр. 22-Рз Наумов А.А.

Проверил:
к.т.н., доцент кафедры ИРС Сидоров С.Б.

Нижний Новгород,
2024 г.

Оглавление

Контрольная работа № 1.
«Реализация индуктивной обработки последовательности элементов»

1.1. Постановка задачи

Освоение способов разработки алгоритмов выполнения индуктивных операций над последовательностью данных, построение индуктивных функций методом индуктивных расширений. Изучение общей схемы программной реализации индуктивной функции и схемы обработки последовательности элементов с использованием индуктивной функции.

На вход системы последовательно и неограниченно во времени поступают элементы xi, где i — порядковый номер элемента, начиная с 1. Реализовать указанную в варианте обработку $f(X)$ последовательности элементов $X = \langle x1, x2, x3, \ldots \rangle$ с использованием схемы индуктивной обработки на пространстве последовательностей. Полученный набор выходных значений $Y = \langle y1, y2, y3, \ldots \rangle$ рассматривается как результирующая последовательность. Значения элементов исходной последовательности должны запрашиваться у пользователя (приниматься со стандартного устройства ввода) по одному за раз. Сформированные выходные значения требуется выдавать сразу после их формирования на стандартное устройство вывода. Реализация обработки должна быть приведена в отдельном модуле.

Задание по варианту

Обнаружение наиболее длинного участка монотонного возрастания значений последовательных элементов, при условии что разность значений последнего и первого элементов участка не менее чем D. Результатом является интервал: (начальный номер, конечный номер). Тип элемента — целочисленный.

1.2. Алгоритм индуктивной обработки

Примем следующие сокращения, и каким значением инициализируются (если значение явно не указывается, то его упоминание пропускается):

tLi – индекс левого элемента, обрабатываемой в данный момент последовательности;

tLv – значение левого элемента, обрабатываемой в данный момент последовательности;

tRi – индекс правого элемента, обрабатываемой в данный момент последовательности;

tRv – значение правого элемента, обрабатываемой в данный момент последовательности;

tCS (*tempCS*)– количество элементов в текущей, обрабатываемой последовательности;

fLi – индекс левого элемента, максимальной по кол-ву элементов последовательности, при разнице значений начального и конечного, на данный момент, элемента, больше, чем D;

fLv – значение левого элемента, максимальной по кол-ву элементов последовательности, при разнице значений начального и конечного, на данный момент, элемента, больше, чем D;

fRi – индекс правого элемента, максимальной по кол-ву элементов последовательности, при разнице начального и конечного, на данный момент, элемента, больше, чем D;;

fRv – значение правого элемента, максимальной по кол-ву элементов последовательности, при разнице начального и конечного, на данный момент, элемента, больше, чем D;

fCS (*finalCS*)– максимальное количество элементов, для обработанных последовательностей, на данный момент.

x_n - очередное поступившее значение;

n - индекс поступившего элемента;

isFirst - булевое значение, вводится как *true*;

lx - предыдущее поступившее значение;

Δv - разница значений первого и последнего элемента максимальной, подходящей, на данный момент, последовательности, $x_n - fLv$;

Δi - разница индексов первого и последнего элемента максимальной, подходящей, на данный момент, последовательности, $n - fCS$;

D – цифра, вводимая пользователем, по условию задачи.

y - элемент выходной последовательности, представлен набором из двух элементов $y = (fLi, fRi)$

Отклик вычислителя *q* определим как $q = \begin{Bmatrix} < y > & , fRv \neq 0 \wedge fLv \neq 0 \\ < \ > & , otherwise \end{Bmatrix}$.

Рассмотрим базовые условия:

$r_1(isFirst)$;

$r_2(x_n \le lx)$;

$r_3(tCS > fCS)$;

$r_4(\Delta v > D)$;

Предикаты:

$R_1() : r_1$

Результат: $R_1(.) \to fLi = 0,\ fRi = 0,\ fLv = 0,\ fRv = 0,\ q = <>,\ isFirst = false$;

$R_2() : \neg r_1 \wedge r2$

Результат: $R_2(.) \to tCS = 0,\ tLi = n,\ tLv = x_n,\ tRi = n,\ tRv = x_n,$
$\Delta v = 0,\ \Delta i = 0,\ q = <>,\ lx = x_n,\ n = n+1$;

$R_3() : \neg r_1 \wedge \neg r_2 \wedge \neg r3$

Результат: $R_3(.) \to tCS = tCS+1,\ tRi = n,\ tRv = x_n,$
$\Delta v = x_n - tLv,\ \Delta i = n - tCS,\ q = <>,\ lx = x_n,\ n = n+1;$;

$R_4() : \neg r_1 \wedge \neg r_2 \wedge r_3 \wedge \neg r_4$

Результат: $R_4(.) \to tCS = tCS+1,\ tRi = n,\ tRv = x_n,$
$\Delta x = x_n - tLv,\ \Delta i = n - tCS,\ q = <>,\ lx = x_n,\ n = n+1$;

$R_5() : \neg r_1 \wedge \neg r_2 \wedge r_3 \wedge r_4$

Результат: $R_5(.) \to tCS = tCS+1,\ tRi = n,\ tRv = x_n,\ fLi = tLi,\ fLv = tLv,\ fRi = n,\ fRv = x_n,$
$fCS = tCS,\ \Delta x = x_n - tLv,\ \Delta i = n - tCS,\ q = <y>,\ lx = x_n,\ n = n+1$

Предикат $R_1(.)$ соответствует случаю, когда программа находится на первой итерации. $R_2(.)$, когда итерация не является первой, и текущее, поступившее значение меньше предыдущего. $R_3(.)$, если итерация не является первой, поступившее значение больше предыдущего, и кол-во элементов в текущей последовательности больше, чем в предыдущей, максимально достигнутой вычислителем. $R_4(.)$, в случае, если итерация не является первой, текущее значение *x* больше предыдущего, кол-во элементов в текущей последовательности больше, чем прежде, максимально достигнутое, и разница первого и последнего элемента **меньше** D. $R_5(.)$, когда итерация не является первой, текущее значение *x* больше предыдущего, кол-во элементов в текущей последовательности больше, чем прежде, максимально достигнутое, и разница первого и последнего элемента **больше** D.

Приведённые условия являются предикатами, т.е. результатом каждого из условий является логическое значение истина/ложь. Точкой обозначен набор аргументов, необходимых для вычисления значения предиката.

В ходе выполнения программы, значение счётчика *tempCS*, претерпевает изменения, в результате индуктивной операции над x_n, соответственно работа счётчика будет считаться индуктивным расширением.

Условия, используемые в правиле пересчёта *fLi* и *fRi* охватывают все возможные ситуации, то есть $R_1 \cup R_2 \cup R_3 \cup R_4 \cup R_5 = true$. Таким образом одно из условий должно обязательно выполняться. В противном случае для некоторых ситуаций отсутствует правило пересчёта величины.

Кроме того, выполняется условие $\forall x_n : R_5 \rightarrow R_1 \cap R_2 \cap R_3 \cap R_4 \cap R_5 = false$, то есть не допускается одновременное выполнение различных условий. И обратное утверждение, $\neg \forall x_n : R_5 \rightarrow R_1 \cap R_2 \cap R_3 \cap R_4 \cap R_5 = true$. Истинное значение предиката обозначает факт наступления связанного с ним события.

1.3. Архитектура программной реализации вычислителя.

Разберём используемый абстрактный тип данных.

АТД Application

Определяется структурный тип данных **`Application`**, содержащий следующие поля:

```
int tLi;
int tLv;
int tRi;
int tRv;
int fLi;
int fRi;
int tCS;
int fCS;
bool isFirst;
int n;
int D;
int xn;
```

Переменные по условию из пункта 1.2.

application.h

В заголовочном файле application.h объявляется глобальная функция int appRun(Application& app). Эта функция отвечает за исполнение работу программы, а соответственно, напрямую взаимодействует с АТД Application, а именно: получает данные от пользователя, обрабатывает их, и выводит результат в стандартное устройство вывода. При успешном выполнении возвращает значение 0, если произошла ошибка на одном из этапов, возвращает значение 1.

Для выполнения поставленных, условием, задач, в `.h` файле объявляется прототип функции, а в `.cpp` файле, определяются под-функции `appRun`, а именно:

```
bool app_get_D(Application &app);
bool app_get_element(Application &app);
```

```
bool app_process(Application &app);
bool app_output_match(Application &app);
```

Рассмотрим каждую функцию в отдельности:

`bool app_get_D(Application &app);` - Получение числа D.

`bool app_get_element(Application &app);`- отвечает за получение очередного значения, от пользователя.

`bool appProcess(Application &app);` - Содержит основную логическую функцию обработки.

`bool app_output_match(Application &app);` - выводит промежуточный результат на стандартное устройство вывода.

Sample:

1 2 3 4 5 1 2 3 4 5 6 7 8 9 10 1 2 3 4 5 6 7 8 9 10 11 12 1 2 3

ASSERT {15, 27}

OUTPUT {15, 27}

Приложение 1.

```
//main.cpp
#include "Application.h"

int main() {

    Application app;
    int ret = app_run(app);

    return ret;
}


//application.h
#ifndef APPLICATION_H
#define APPLICATION_H

struct Application {
    int tLi;
    int tLv;
    int tRi;
    int tRv;
    int fLi;
    int fRi;
    int tCS;
    int fCS;
    bool isFirst;
    int n;
    int D;
    int xn;

    Application(): tLi(0), tLv(0), tRi(0), tRv(0), fLi(0), fRi(0),
    tCS(1), fCS(1), isFirst(true), n(0), D(0), xn(0) {}
};

bool app_get_D(Application &app);
bool app_get_element(Application &app);
bool app_output_match(Application &app);
bool app_process(Application &app);
int app_run(Application &app);

#endif // APPLICATION_H
```

```cpp
//application.cpp
#include <iostream>
#include "Application.h"

bool app_get_D(Application &app) {
    std::cout << "Enter the value of D: ";
    std::cin >> app.D;
    std::cout << app.D << std::endl;
    return true;
}

bool app_get_element(Application &app) {
    std::cin >> app.xn;
    return true;
}

bool app_output_match(Application &app) {
    if (app.fLi != 0 && app.fRi != 0) {
        std::cout << "Current best segment: from " << app.fLi + 1 << " to " << app.fRi + 1 <<
std::endl;
    }
    return true;
}

bool app_process(Application &app) {
    if (app.isFirst) {
        app.tLv = app.tRv = app.xn;
        app.isFirst = false;
    }
    if (app.tCS == 1) {
        app.tLi = app.n - 1;
        app.tLv = app.tRv;
        app.tRi = app.n;
        app.tRv = app.xn;
        app.tCS = 2;
    } else {
        if (app.xn > app.tRv) {
            app.tRi = app.n;
            app.tRv = app.xn;
            app.tCS++;
        } else {
            app.tLi = app.n;
            app.tLv = app.xn;
            app.tRi = app.n;
            app.tRv = app.xn;
            app.tCS = 1;
```

```
        }
        if (app.tCS > app.fCS && (app.tRv - app.tLv >= app.D)) {
            app.fLi = app.tLi;
            app.fRi = app.tRi;
            app.fCS = app.tCS;
        }
    }
    app.n++;
    return true;
}

int app_run(Application &app) {
    app_get_D(app);
    while (!std::cin.eof()) {
        app_get_element(app);
        app_process(app);
        app_output_match(app);
    }
    return 0;
}
```

Контрольная работа №2
«Настройка индуктивного вычислителя с использованием функции обратного вызова»

2.1 Архитектура программной системы

Основная логика вычисления не претерпела изменений, с контрольной работы 1. Изменился стиль вызова и способ обмена данными между функциями. Теперь мы объявляем АТД Application в основной функции `app_Run`, вместо main.

В следствие внедрения данной методологии вызова функций, изменилась структура программы. Представлена ниже.

2.2 Использование индуктивного вычислителя

Функция operation, которая соответствует назначению — callback из условия работы. Она принимает ссылку на функцию и ссылку на объявленный тип данных. В результате мы избегаем копирования одного и того же типа данных, по несколько раз.

В следствие использования данной структуры, можно наладить общение нескольких программ между друг другом, вследствие унифицированного элемента функции и набора данных.

//application.h

```
typedef bool (*Callback)(void *object);
bool operation(Callback callback, void *data);
```

Элемент	Назначение
`typedef`	аллиас на функцию
`bool`	тип функции
`(*Callback)`	указатель на тип функции
`(void *object)`	указатель на тип принимаемого аргумента, и его название

```
// application.cpp
bool operation(Callback callback, void *data) {
    return (*callback)(data);
}
```

Элемент	Назначение
`bool`	тип функции
`operataion`	название
`(Callback callback`	тип и имя принимаемого аргумента
`void *data)`	указатель на любой тип данных, имя аргумента
`return`	возвращаемое значение
`(*callback)`	результат работы вызываемой функции
`(data)`	данные с которыми работала программа.

Приложение 1

```
//main.cpp
#include "Application.h"

int main() {

    Application app;
    int ret = app_run(&app);

    return ret;
}

//application.h
#ifndef APPLICATION_H
#define APPLICATION_H

struct Application {
    int tLi;
    int tLv;
    int tRi;
    int tRv;
    int fLi;
    int fRi;
    int tCS;
    int fCS;
    bool isFirst;
    int n;
    int D;
    int xn;

    Application(): tLi(0), tLv(0), tRi(0), tRv(0), fLi(0), fRi(0),
    tCS(1), fCS(1), isFirst(true), n(0), D(0), xn(0) {}
};

typedef bool (*Callback)(void *object);
bool operation(Callback callback, void *data);

bool app_get_D(void* object);
bool app_get_element(void* object);
bool app_output_match(void* object);
bool app_process(void* object);
int app_run(void* object);

#endif // APPLICATION_H
```

```cpp
//application.cpp
#include <iostream>
#include "application.h"

bool operation(Callback callback, void *data) {
    return (*callback)(data);
}

bool app_get_D(void* object) {
    Application &app = *((Application*)object);
    std::cout << "Enter the value of D: ";
    std::cin >> app.D;
    std::cout << app.D << std::endl;
    return true;
}

bool app_get_element(void* object) {
    Application &app = *((Application*)object);
    std::cin >> app.xn;
    return true;
}

bool app_output_match(void* object) {
    Application &app = *((Application*)object);
    if (app.fLi != 0 && app.fRi != 0) {
        std::cout << "Current best segment: from " << app.fLi + 1 << " to " << app.fRi + 1 <<
std::endl;
    }
    return true;
}

bool app_process(void* object) {
    Application &app = *((Application*)object);
    if (app.isFirst) {
        app.tLv = app.tRv = app.xn;
        app.isFirst = false;
    }
    if (app.tCS == 1) {
        app.tLi = app.n - 1;
        app.tLv = app.tRv;
        app.tRi = app.n;
        app.tRv = app.xn;
        app.tCS = 2;
    } else {
        if (app.xn > app.tRv) {
            app.tRi = app.n;
```

```
            app.tRv = app.xn;
            app.tCS++;
        } else {
            app.tLi = app.n;
            app.tLv = app.xn;
            app.tRi = app.n;
            app.tRv = app.xn;
            app.tCS = 1;
        }
        if (app.tCS > app.fCS && (app.tRv - app.tLv >= app.D)) {
            app.fLi = app.tLi;
            app.fRi = app.tRi;
            app.fCS = app.tCS;
        }
    }
    app.n++;
    return true;
}

int app_run(void* object) {
    Application &app = *((Application*)object);
    app_get_D(&app);
    while (!std::cin.eof()) {
        app_get_element(&app);
        app_process(&app);
        app_output_match(&app);
    }
    return 0;
}
```