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 41)

Выполнил:
Студент гр. 22-Рз _____

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

Нижний Новгород,
2024 г.

Оглавление

Задание по варианту

На вход поступают элементы последовательности в виде точек на декартовой плоскости. Для каждого элемента рассматривается группа не менее чем K соседних элементов, включающая его самого и предшествующих ему. Для каждого из элементов этой группы должно быть выполнено условие — точка находится внутри области, ограниченной заданной окружностью. Определить факт наличия такой группы в текущий момент времени и ее размер (количество элементов).

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.

1.2. Алгоритм индуктивной обработки

Алгоритм программы заключается в обработке групп чисел, соответственно, логически, он разбит на несколько функций. По условию, мы обрабатываем точки в декартовой системе координат, следовательно мы принимаем два значения за раз. Договоримся, что x_n - очередная пара точек x/y. Таким образом, для пересчета очередного поступившего значения, должно наступить событие:

$R():r_1 \cap r_2$

Будем использовать следующие сокращения:

r_c	Радиус заданной окружности
r_x	Радиус от центра окружности, до указанной точки

i	Итератор от 0 до k

Можем получить следующие базовые условия обработки:

$r_1(i-1=k)$;

$r_2(r_{xn}<r_c \wedge ... \wedge r_{\text{xn+k}}<r_c)$;

Имея набор базовых условий, можно сформировать предикаты,

$R_1():r_1$ - справедливо для $i-1=k$;

Результат: $R_1(.) \to R_2(.)$;

Обратный предикат $R_2():\neg r_1$, справедлив в случае $i<k$, тогда результат будет:

$R_2(.) \to ++i$;

Предикат 3:

$R_3():r_1 \wedge \neg r_2$ - справедливо для $i-1=k \wedge r_{xn} \geq r_c$в таком случае, результат:

$R_3(.) \to ++i$;

Предикат 4:

$R_4():r_1 \wedge r_2$ - справедливо для $i-1=k \wedge r_{xn}<r_c \wedge \ldots \wedge r_{\text{xn+k}}<r_c$;

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

Для выполнения поставленной задачи, в ходе программы, счётчик i будет проходить значения от 0 до $k-1$, что в свою очередь будет являться индуктивным расширением.

Используемые условия, охватывают все возможные ситуации, то есть $R_1 \cup R_2 \cup R_3 \cup R_4 = true$. Таким образом одно из условий должно обязательно выполняться. В противном случае для некоторых ситуаций отсутствует правило пересчёта величины.

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

1.3. Архитектура программной реализации вычислителя.

Для хранения переменных, используемых в программе, мы используем структуру АТД.

АТД Application

Определяется структурный тип данных **`Application`**, содержащий семь полей:

```
1) std::vector<std::pair <bool, std::pair<double, std::pair<int, int>>>> temp_group;
2) std::pair <int, int>init_xy;

3) std::pair<int,int> circle_center;
4) std::pair<int,int> circle_edge;
5) int circle_r;
6) int const_k;
7) int iteration = 1;
```

Первое — вектор, по размеру k, тип данных которого пара: *bool* и пара *double* и пара int int. Можем рассмотреть данный тип данных с обратной стороны. На низшем уровне — это пара x/y для рассматриваемой точки. На уровень выше, в первое значение пары становится радиус от центра окружности до точки. Поднимаясь еще на уровень выше, в первое значение *bool* – изначально *false*. После обработки, он может быть изменен на *true*.

Второе — пара изначальных значений x/y.

Третье — пара x/y для центра окружности.

Четвертое — точка на окружности, для определения радиуса референсной окружности.

Пятое — вычисленный радиус такой окружности.

Шестое — введенное число k.

Седьмое — счетчик итерации.

application.h

В файле объявляется основная функция рассматриваемого приложения appRun(), которая отвечает за вызов других под-функций, соответственно за обработку поступающих значений x_n.

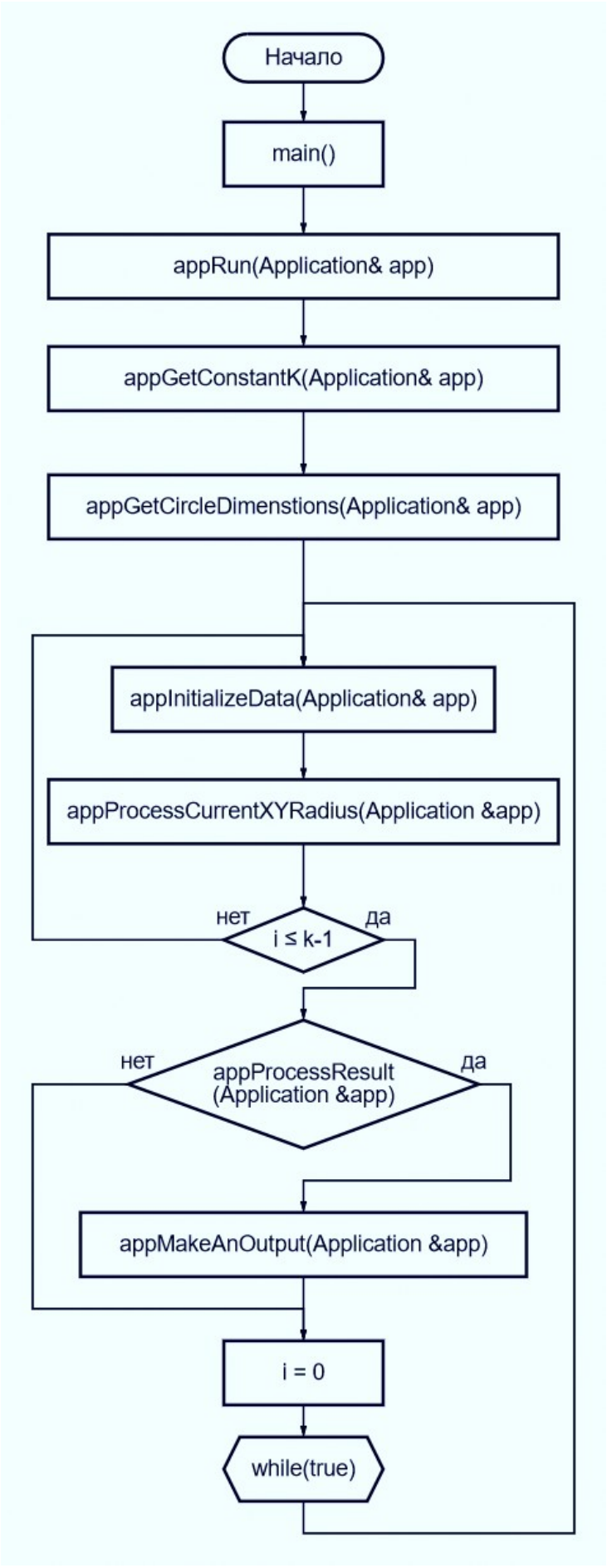
В качестве функций, в `.h` файле объявляется прототип функции, а в `.cpp` файле приводится определение.

```
bool appGetConstantK(Application &app);
bool appGetCircleDimensions(Application &app);
bool appInitializeData(Application &app);
bool appProcessCurrentXYRadius(Application &app);
bool appProcessResult(Application &app);
bool appGetOutputToUser(Application &app);
```

Рассмотрим каждую функцию в отдельности:

`bool appGetConstantK(Application &app);` - Получение числа K;
`bool appGetCircleDimensions(Application &app);` - Получение center и edge координат, референсной окружности;
`bool appInitializeData(Application &app);` - Считывание очередной пары координат;
`bool appProcessCurrentXYRadius(Application &app);` - единичная функция сборки вектора, по размеру k;
`bool appProcessResult(Application &app);` - проверка вхождения точек группы в референсную окружность;
`bool appGetOutputToUser(Application &app);` - выдача результата пользователю.

Можем представить общую структуру программы в виде блок-схемы:


Начало
main()
appRun(Application& app)
appGetConstantK(Application& app)
appGetCircleDimenstions(Application& app)
appInitializeData(Application& app)
appProcessCurrentXYRadius(Application &app)
нет
i ≤ k-1
да
нет
appProcessResult
(Application &app)
да
appMakeAnOutput(Application &app)
i = 0
while(true)

Приложение 1.

Main.cpp

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

int main() {

    std::cout << "main.cpp" << std::endl;

    Application app;

    int ret = appRun(app);

    return ret;
}
```

application.h

```
#pragma once
#ifndef NNTU_APPLICATION_H
#define NNTU_APPLICATION_H

#include <climits>
#include <utility>
#include <vector>

struct Application {

    //assume 1 - bool of whether radius belongs to the target
    //          2 - 1- Computed R of a given set of XY,
    //              2 - initial XY
    std::vector<std::pair <bool, std::pair<double, std::pair<int, int>>>>
temp_group;
    std::pair <int, int>init_xy;

    //CIRCLE
    //Assume 1-X 2-Y
    std::pair<int,int> circle_center;
    std::pair<int,int> circle_edge;
```

```
    int circle_r;
    int const_k;
    int iteration = 1;
};

int appRun(Application& app);

bool appGetConstantK(Application &app);
bool appGetCircleDimensions(Application &app);
bool appInitializeData(Application &app);
bool appProcessCurrentXYRadius(Application &app);
bool appProcessResult(Application &app);
bool appGetOutputToUser(Application &app);

#endif //NNTU_APPLICATION_H


#include "application.h"
#include <iostream>
#include <cmath>

int appRun(Application &app) {
    if (!appGetConstantK(app)) {
        std::cout << "DATA INPUT FAILURE." << std::endl;
        return 1;
    }

    if (!appGetCircleDimensions(app)) {
        std::cout << "DATA INPUT FAILURE." << std::endl;
        return 1;
    }

    while (!std::cin.eof()) {
        int i;
        while (i <= app.const_k) {
            if (i <= app.const_k - 1) {
                if (!appInitializeData(app)) {
                    std::cout << "DATA INPUT FAILURE." << std::endl;
                    return 1;
                }
```

```
            if (!appProcessCurrentXYRadius(app)) {
                std::cout << "DATA INPUT FAILURE." << std::endl;
                return 1;
            }
        }
        else {
            if (appProcessResult(app)) {
                appGetOutputToUser(app);
            }
            app.temp_group.clear();
        }
        ++i;
    }
    ++app.iteration;
    i = 0;
  }
  return 0;
}

bool appGetConstantK(Application &app) {
  std::cout << "Input a K constant:" << std::endl;
  std::cin >> app.const_k;
  std::cout << app.const_k << std::endl;
  return true;
}

bool appGetCircleDimensions(Application &app) {
  std::cout << "Input CENTER 'X Y' coordinate of circle: " << std::endl;
  std::cin >> app.circle_center.first >> app.circle_center.second;
  std::cout << app.circle_center.first <<"/"<< app.circle_center.second <<
std::endl;

  std::cout << "Input EDGE 'X Y' coordinate of circle: " << std::endl;
  std::cin >> app.circle_edge.first >> app.circle_edge.second;
  std::cout << app.circle_edge.first <<"/"<< app.circle_edge.second <<
std::endl;

  app.circle_r = sqrt(
        pow(app.circle_edge.first - app.circle_center.first, 2) +
```

```
            pow(app.circle_edge.second - app.circle_center.second, 2));

    return true;
}

bool appInitializeData(Application &app) {

    std::cin >> app.init_xy.first >> app.init_xy.second;

    return true;
}

bool appProcessCurrentXYRadius(Application &app) {

    app.temp_group.push_back(std::make_pair(false,std::make_pair(sqrt(
        pow(app.init_xy.first - app.circle_center.first, 2) +
        pow(app.init_xy.second - app.circle_center.second, 2)),
         std::make_pair(app.init_xy.first, app.init_xy.second))));

    return true;
}


bool appProcessResult(Application &app) {

    int counter = 0;

    for (int j = 0; j < app.temp_group.size(); ++j) {
        if (app.temp_group[j].second.first < app.circle_r) {
            app.temp_group[j].first = true;
            ++counter;
        }
    }

    if (counter == app.const_k){
        return true;
    }

    return false;
}
```

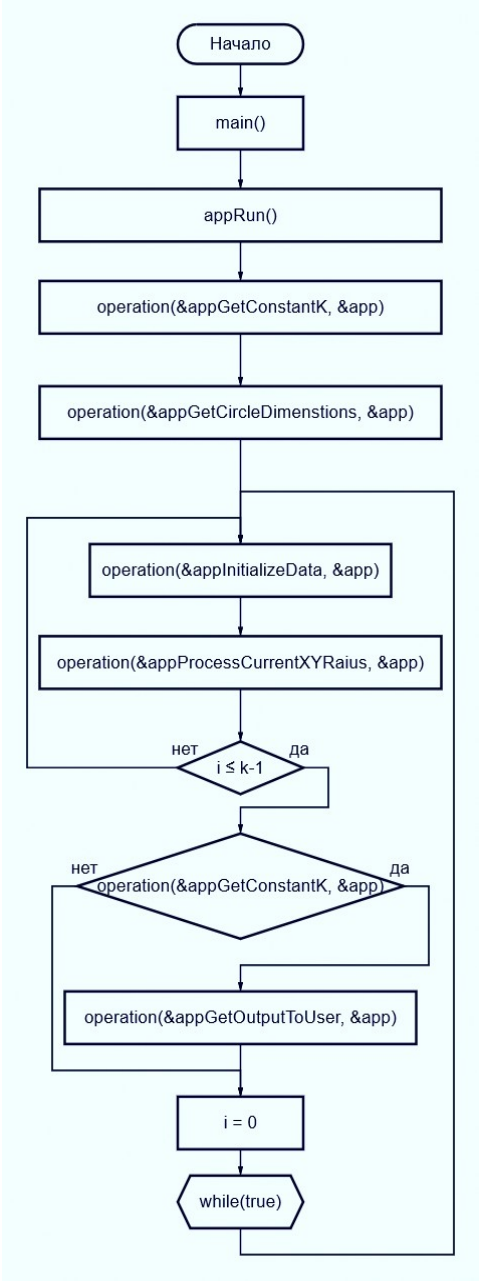
```
bool appGetOutputToUser(Application &app) {

    std::cout << app.iteration << " Iteration - group is" << "inside the circle, their values are:" << std::endl;

    for (int i = 0; i < app.const_k; ++i) {

        std::cout << "(" << app.temp_group[i].second.second.first << "/" << app.temp_group[i].second.second.second << ") ";

    }
    std::cout << std::endl;
    return true;
}
```

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

2.1 Архитектура программной системы

Алгоритм обработки поступающих значений не претерпел изменений, но был видоизменён стиль вызова функций, и передачи им данных для работы. Теперь мы используем функцию *callback*, передаём ей в качестве аргументов, ссылку на функцию для запуска, и ссылку на данные, с которыми предстоит работать.

Логично предположить, что исходя из внесённых изменений, немного изменилась структура программы, обновлённая версия, в виде блок-схемы, представлена ниже.



2.2 Использование индуктивного вычислителя

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

```
//application.h

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

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

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

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
`(ADT)`	данные с которыми работала программа.

Приложение 1

main.cpp

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

int main() {

    std::cout << "main.cpp" << std::endl;

    int ret = appRun();

    return ret;
}
```

application.h

```
#include <climits>
#include <utility>
#include <vector>

struct Application {

    //assume 1 - bool of whether radius belongs to the target
    //          2 - 1- Computed R of a given set of XY,
    //              2 - initial XY
    std::vector<std::pair <bool, std::pair<double, std::pair<int, int>>>>
temp_group;
    std::pair <int, int>init_xy;

    //CIRCLE
    //Assume 1-X 2-Y
    std::pair<int,int> circle_center;
    std::pair<int,int> circle_edge;
    int circle_r;
    int const_k;
    int iteration = 1;
};

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

```
int appRun();

bool appGetConstantK(void *object);
bool appGetCircleDimensions(void *object);
bool appInitializeData(void *object);
bool appProcessCurrentXYRadius(void *object);
bool appProcessResult(void *object);
bool appGetOutputToUser(void *object);

#endif //NNTU_APPLICATION_H
```

application.cpp

```
#include "application.h"
#include <iostream>
#include <cmath>

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

bool appGetConstantK(void *object) {

    Application &app = *(Application*) object;

    std::cout << "Input a K constant:" << std::endl;
    std::cin >> app.const_k;
    std::cout << app.const_k << std::endl;
    return true;
}

bool appGetCircleDimensions(void *object) {

    Application &app = *(Application*) object;

    std::cout << "Input CENTER 'X Y' coordinate of circle: " << std::endl;
    std::cin >> app.circle_center.first >> app.circle_center.second;
    std::cout << app.circle_center.first <<"/"<< app.circle_center.second <<
std::endl;
```

```
    std::cout << "Input EDGE 'X Y' coordinate of circle: " << std::endl;
    std::cin >> app.circle_edge.first >> app.circle_edge.second;
    std::cout << app.circle_edge.first <<"/"<< app.circle_edge.second <<
std::endl;

    app.circle_r = sqrt(
            pow(app.circle_edge.first - app.circle_center.first, 2) +
            pow(app.circle_edge.second - app.circle_center.second, 2));

    return true;
}

bool appInitializeData(void *object) {

    Application &app = *(Application*) object;

    std::cin >> app.init_xy.first >> app.init_xy.second;

    return true;
}

bool appProcessCurrentXYRadius(void *object) {

    Application &app = *(Application*) object;

    app.temp_group.push_back(std::make_pair(false,std::make_pair(sqrt(
                                pow(app.init_xy.first - app.circle_center.first, 2) +
                                 pow(app.init_xy.second - app.circle_center.second, 2)),
                                 std::make_pair(app.init_xy.first, app.init_xy.second))));

    return true;
}

bool appProcessResult(void *object) {

    Application &app = *(Application*) object;

    int counter = 0;

    for (int j = 0; j < app.temp_group.size(); ++j) {
```

```
        if (app.temp_group[j].second.first < app.circle_r) {
            app.temp_group[j].first = true;
            ++counter;
        }
    }

    if (counter == app.const_k){
        return true;
    }

    return false;
}

bool appGetOutputToUser(void *object) {

    Application &app = *(Application*) object;

    std::cout << app.iteration << " Iteration - group is" << "inside the circle, their values are:" << std::endl;

    for (int i = 0; i < app.const_k; ++i) {

        std::cout << "(" << app.temp_group[i].second.second.first << "/" << app.temp_group[i].second.second.second << ") ";

    }
    std::cout << std::endl;
    return true;
}

int appRun() {

    Application app;

    if (!operation(&appGetConstantK, &app)) {
        std::cout << "DATA INPUT FAILURE." << std::endl;
        return 1;
    }

    if (!operation(&appGetCircleDimensions, &app)) {
```

```
        std::cout << "DATA INPUT FAILURE." << std::endl;
        return 1;
    }

    while (!std::cin.eof()) {
        int i;
        while (i <= app.const_k) {
            if (i <= app.const_k - 1) {
                if (!operation(&appInitializeData, &app)) {
                    std::cout << "DATA INPUT FAILURE." << std::endl;
                    return 1;
                }

                if (!operation(&appProcessCurrentXYRadius, &app)) {
                    std::cout << "DATA INPUT FAILURE." << std::endl;
                    return 1;
                }
            }
            else {
                if (operation(&appProcessResult, &app)) {
                    operation(&appGetOutputToUser, &app);
                }
                app.temp_group.clear();
            }
            ++i;
        }
        ++app.iteration;
        i = 0;
    }
    return 0;
}
```