МИНОБРНАУКИ РОСИИ

САНКТ_ПЕТЕРБУРГСКИЙ ГОСУДАРСВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

<<ЛЭТИ>> ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра МО ЭВМ

Тема лабораторной работы:

Реализация и исследование развёрнутого связного списка

Студент гр. 2300

Хидда Абдула

Санк-Петербург

2023

Цель работы.

Реализовать развёрнутый связный список, сравнить его скорость работы с другими структурами данных.

Задание.

1) Реализовать класс развёрнутого связного списка согласно описанию.
2) Реализовать функцию проверки check.
3) Исследовать сравнить время работы с обычным массивом и списком.

Выполнение работы.

Реализован класс block:

int max_bl_size – максимальный размер блока логической информации

int bl_end – размер записанной информации

int* bl – указатель на блок логической информации

class block* next – указатель на следующий блок информации

Созданы его конструкторы с задаваемым значением длины массива логической информации и дефолтным (определена директивой define)

Реализован класс ex_lin_list:

Класс является нашим развёрнутым связным списком (на графиках будет обозначен как e_l_list)

int list_size – количество элементов в списке

class block* first – указатель на первый блок

class block* find_bl(int& index) – функция находящая блок в котором хранится элемент с данным индексом и изменяющая значение индекса на то, которым он обладает в этом блоке

int find_val(int index) – функция которая возвращает значение элемента по идексу

int del_val(int index) – функция удаляющая элемент по индексу

int insert_val(int index, int val) – функция вставляющая элемент в заданный индекс с заданным значением

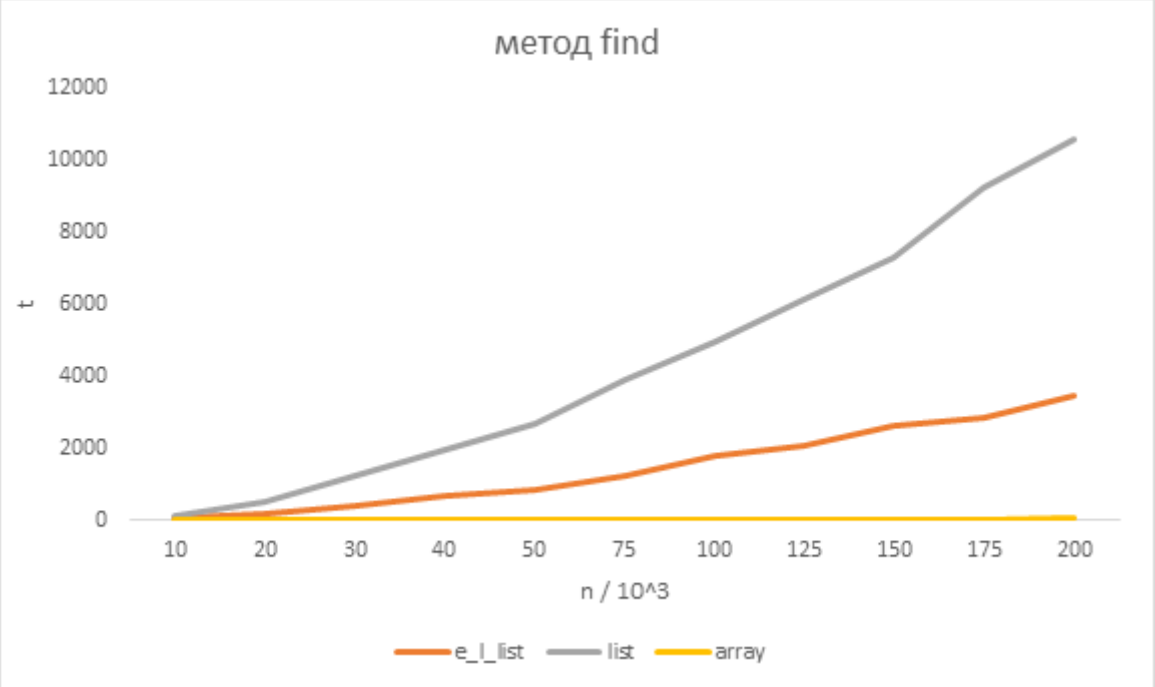
Созданы его конструкторы с задаваемым значением длины массива логической информации и дефолтным, создан деструктор.

Тесты

(развёрнутый связный список отмечен как e_l_list, во всех тестах использовалась его дефолтная версия с размером блока равным 4, для подсчёта времени использовалась функция clock() библиотеки time.h)

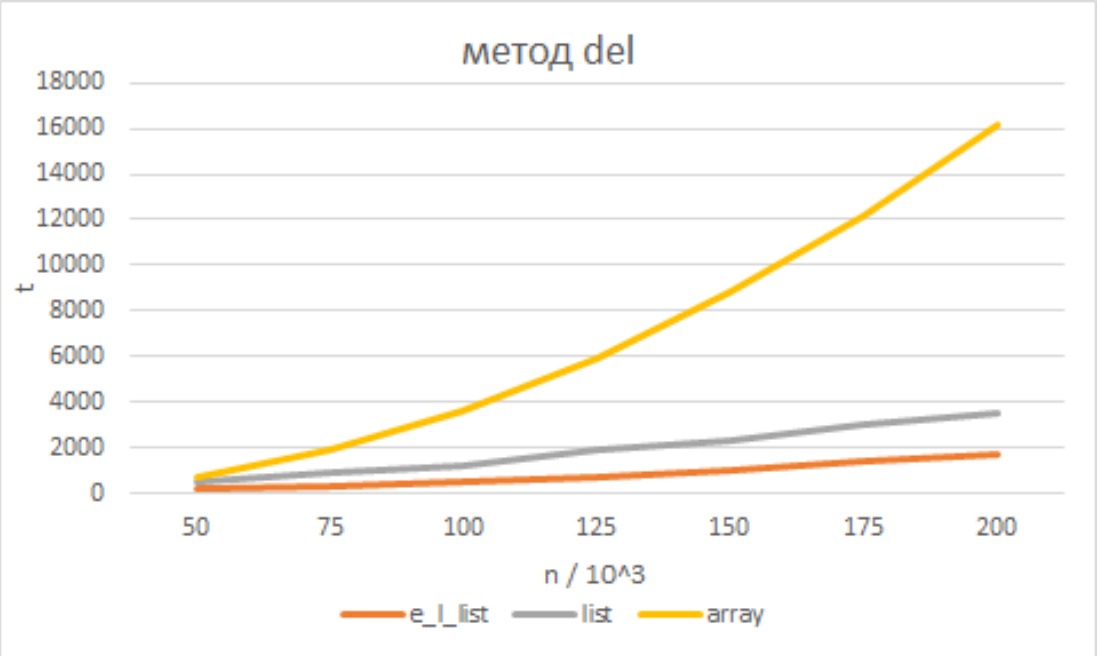
В результате тестирования функции find_val, которая ищет значение по индексу получены данные:

n/10^3	e_l_list	list	array
10	41	117	0
20	148	491	0
30	419	1234	0
40	659	1955	0
50	849	2669	0
75	1249	3896	13
100	1801	4942	17
125	2082	6137	25
150	2627	7285	22
175	2862	9218	26
200	3455	10571	40



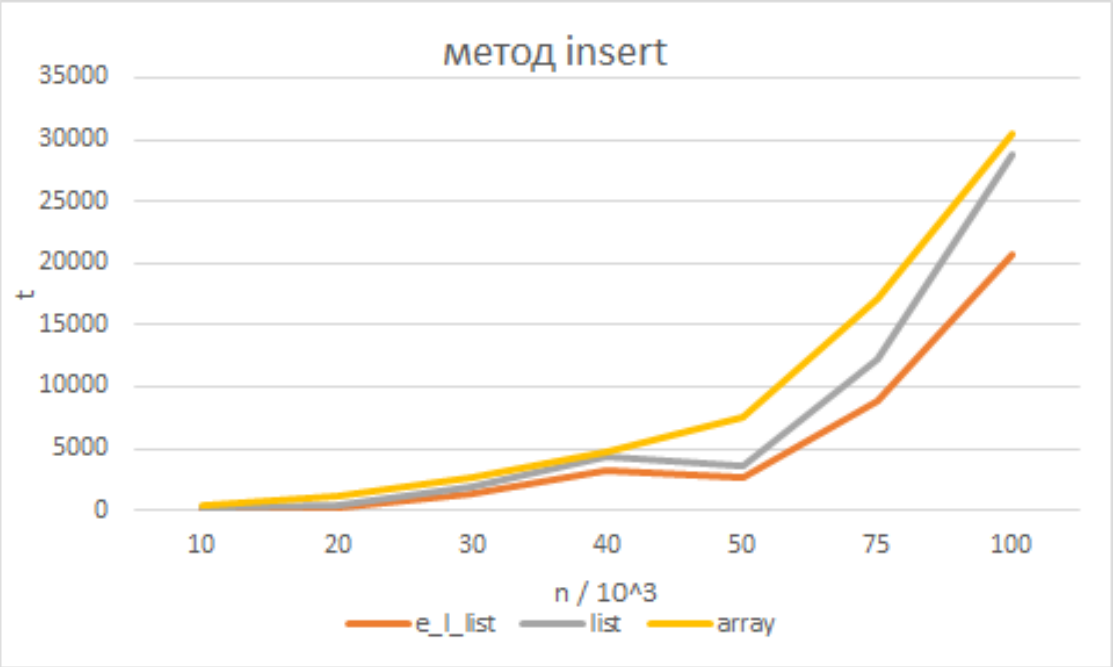
В результате тестирования функции del_val, которая удаляет элемент по индексу получены данные:

n/10^3	e_l_list	list	array
50	228	563	701
75	331	932	1880
100	518	1256	3645
125	762	1884	5920
150	1029	2361	8789
175	1406	2990	12171
200	1680	3501	16147



В результате тестирования функции insert_val, которая вставляет элемент с заданным значением по индексу получены данные:

n/10^3	e_l_list	list	array
10	89	139	390
20	300	445	1224
30	1308	1941	2672
40	3267	4359	4733
50	2704	3511	7545
75	8774	12324	17210
100	20770	28818	30507



Вывод: Развёрнутый связный список работает быстрее чем обычный список и массив (единственное исключение – массив при поиске по индексу, т.к. он работает за O(1)). На операциях поиска и удаления наш развёрнутый список примерно в 2 быстрее обычного. При изменении размера блока возможно можно добиться ещё большего выигрыша во времени.

Приложение

Исходный код программы

Название файла: main.cpp

```
#include <iostream>
#include "ex_lin_list.h"
#include <math.h>

#include <time.h>
#include <fstream>
#include <string>

#include "list.cpp"

using namespace std;

void check(int* arr_1, int n_1, int arr_2, int n_2, int n_array);

int main(){
   class ex_lin_list A;

   int N_TEST = 10000;


   std::ifstream in1;
   string s1 = "tests/fill" + to_string(N_TEST) + ".txt";
   in1.open(s1);
   for (int i = 0; i < N_TEST; i++){
      int a;
      in1 >> a;
      A.insert_val(i, a);
   }

   clock_t startTime, endTime;
   startTime = clock( );
//CODE TEST
//-----------------------------------------------
//find test

   std::ifstream in2;
   string s2 = "tests/find" + to_string(N_TEST) + ".txt";
   in2.open(s2);
   for (int i = 0; i < N_TEST; i++) {
      int a;
```

```
        in2 >> a;
        A.find_val(a);
    }

//del test
/*
    std::ifstream in2;
    string s2 = "tests/find" + to_string(N_TEST) + ".txt";
    in2.open(s2);
    for (int i = 0; i < N_TEST / 5; i++) {
        int a;
        in2 >> a;
        A.del_val(a);
    }
*/
//insert test
/*
    std::ifstream in2;
    string s2 = "tests/find" + to_string(N_TEST) + ".txt";
    in2.open(s2);
    for (int i = 0; i < N_TEST; i++) {
        int a, b;
        in2 >> a >> b;
        A.insert_val(a, b);
    }
*/
//-----------------------------------------------
    endTime = clock( );
    cout << (endTime - startTime);


//GENERATE TESTS
//-----------------------------------------------
/*
    std::ofstream out10;
    string s10 = "tests/fill" + to_string(N_TEST) + ".txt";
    out10.open(s10);
    for (int i = 0; i < N_TEST; i++)
        out10 << rand()%1000 << " ";

    std::ofstream out11;
    string s11 = "tests/find" + to_string(N_TEST) + ".txt";
    out11.open(s11);
    for (int i = 0; i < N_TEST; i++)
        out11 << rand()%N_TEST << " ";

    std::ofstream out12;
    string s12 = "tests/del" + to_string(N_TEST) + ".txt";
    out12.open(s12);
    for (int i = 0; i < N_TEST / 5; i++)
        out12 << rand()%(N_TEST / 5) << " ";

    std::ofstream out13;
    string s13 = "tests/insert" + to_string(N_TEST) + ".txt";
    out13.open(s13);
```

```
   for (int i = 0; i < N_TEST; i++)
      out13 << rand()%N_TEST << " " << rand()%1000 << " ";
*/
//-----------------------------------------------
}

void check(int* arr_1, int n_1, int* arr_2, int n_2, int n_array){
   ex_lin_list A;
   for (int i = 0; i < n_1; i++){
      A.insert_val(i, arr_1[i]);
      A.write_out();
      cout << endl;
   }
   for (int i = 0; i < n_2; i++){
      A.del_val(arr_2[i]);
      A.write_out();
      cout << endl;
   }
}
```

Название файла: block.h

```
#pragma once

#define BL_SIZE_DEFAULT 4
#define BL_SIZE_MIN 1

class block{
    public:
        int max_bl_size;
        int bl_end;
        int* bl;
        class block* next;

        block();
        block(int size_of_bl);
};
```

Название файла: block.cpp

```
#pragma once

#include "block.h"
#include <iostream>

block::block(){
    max_bl_size = BL_SIZE_DEFAULT;
    bl = new int[max_bl_size];
    next = nullptr;

    bl_end = 0;
}

block::block(int size_of_bl){
    if (size_of_bl >= BL_SIZE_MIN)
        max_bl_size = size_of_bl;
```

```
    else
        max_bl_size = BL_SIZE_MIN;
    bl = new int[max_bl_size];
    next = nullptr;

    bl_end = 0;
}
```

Название файла: ex_lin_list.h

```
#pragma once

#include "block.h"

class ex_lin_list{
    private:
        int list_size;
        class block* first;

        class block* find_bl(int& index);
    public:
        ex_lin_list();
        ex_lin_list(int bl_size);
        int find_val(int index);
        void del_val(int index);
        void insert_val(int index, int val);
        void write_out();
        ~ex_lin_list();
};
```

Название файла: ex_lin_list.cpp

```
#pragma once

#include "ex_lin_list.h"
#include <iostream>

ex_lin_list::ex_lin_list(){
    first = new block();
    list_size = 0;
}

ex_lin_list::ex_lin_list(int bl_size){
    first = new block(bl_size);
    list_size = 0;
}

class block* ex_lin_list::find_bl(int& index){
    class block* current_bl = first;
    while (current_bl != nullptr){
        if (current_bl->bl_end > index)
            break;
        index -= current_bl->bl_end;
        current_bl = current_bl->next;
    }

    return current_bl;
};

int ex_lin_list::find_val(int index){
```

```
    if (index >= list_size || index < 0){
        std::cout << "find_val is out of list_size" << std::endl;
        return INT_MIN;
    }

    class block* current_bl = find_bl(index);
    return current_bl->bl[index];
}

void ex_lin_list::del_val(int index){
    if (index >= list_size || index < 0){
        std::cout << "del_val is out of list_size" << std::endl;
        return;
    }

    class block* current_bl = find_bl(index);
    for (int i = index; i + 1 < current_bl->bl_end; i++)
        current_bl->bl[i] = current_bl->bl[i + 1];
    current_bl->bl_end--;
}

void ex_lin_list::insert_val(int index, int val){
    if (index > list_size || index < 0){
        std::cout << "insert_val is out of list_size" << std::endl;
        return;
    }

    index--;
    class block* current_bl = find_bl(index);
    index++;

    if (index < current_bl->max_bl_size && index >= current_bl->bl_end){
        current_bl->bl[index] = val;
        current_bl->bl_end++;
        list_size++;
    }
    else{
        class block* new_bl = new block(current_bl->max_bl_size);
        new_bl->next = current_bl->next;
        current_bl->next = new_bl;

        for (int i = index; i < current_bl->bl_end; i++){
            new_bl->bl[new_bl->bl_end] = current_bl->bl[i];
            new_bl->bl_end++;
        }
        current_bl->bl_end -= new_bl->bl_end;

        if (index == current_bl->max_bl_size){
            new_bl->bl[0] = val;
            new_bl->bl_end++;
        }
        else {
            current_bl->bl[index] = val;
            current_bl->bl_end++;
        }
        list_size++;
    }
}
```

```
void ex_lin_list::write_out(){
    class block* current_bl = first;
    while (current_bl != nullptr){
        for (int i = 0; i < current_bl->bl_end; i++)
            std::cout << current_bl->bl[i] << " ";
        //std::cout << "| ";
        current_bl = current_bl->next;
    }
}

ex_lin_list::~ex_lin_list(){
    class block* current_bl = first;
    while (current_bl != nullptr){
        class block* mem = current_bl->next;
        delete current_bl;
        current_bl = mem;
    }
}
```