**МИНОБРНАУКИ РОСИИ**

**САНКТ\_ПЕТЕРБУРГСКИЙ ГОСУДАРСВЕННЫЙ**

**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**<<ЛЭТИ>> ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра MO 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 сheck.
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

сlass 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

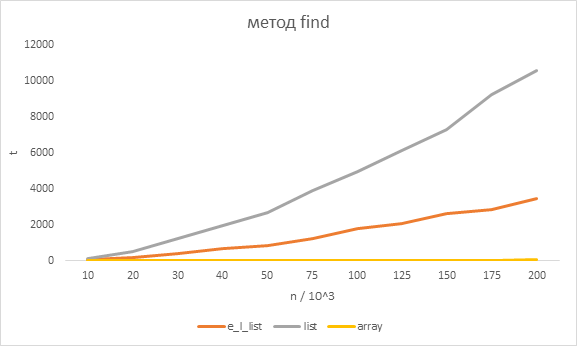
Созданы его конструкторы с задаваемым значением длины массива логической информации и дефолтным, создан деструктор.

**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)

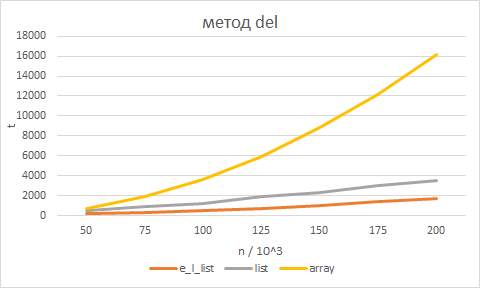
**В результате тестирования функции find\_val, которая ищет значение по индексу получены данные:**

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |



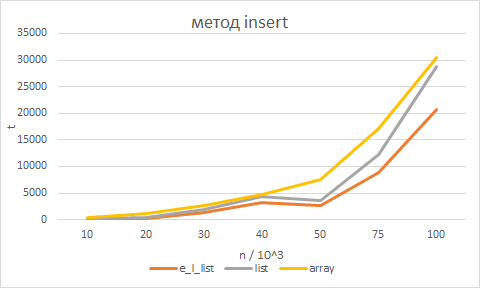
**В результате тестирования функции del\_val, которая удаляет элемент по индексу получены данные:**

|  |  |  |  |
| --- | --- | --- | --- |
| 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;

}

}