**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ**

**УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ**

**ГОМЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ П. О. СУХОГО**

Факультет автоматизированных и информационных систем

Кафедра «Информатика»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6

по дисциплине «Алгоритмы и структуры данных»

на тему: «Архивация данных»

Выполнил: студент гр. ИП-31

Казутин П. Н.

Принял: ст. преподаватель

Косинов Г. П.

Гомель 2021

**Цель работы:** изучить основные методы архивации, сравнить их эффективность.

**Практическая часть:**

Реализовать алгоритмы сжатия согласно варианту. Определить коэффициенты сжатия и время архивации.

|  |  |  |  |
| --- | --- | --- | --- |
| Вариант + доп требования | | Алгоритм 1 | Алгоритм 2 |
| 9 | С паролем | Арифметическое сжатие | RLE+BWT |

#include "list\_chain.h"

list\_chain::list\_chain(std::list<list\_node\_chain\*> nodes)

{

this->head = nodes.front();

this->\_length = nodes.size();

}

void list\_chain::append(list\_node\_chain\* element)

{

if (head == nullptr)

head = element;

else {

list\_node\_chain\* top = head;

while (top->next != nullptr)

top = top->next;

top->next = element;

}

\_length++;

}

void list\_chain::insert(list\_node\_chain\* element, int position\_after)

{

list\_node\_chain\* top = getByNumber(position\_after);

if (top == nullptr) {

append(element);

\_length++;

return;

}

else {

auto tmp = top->next;

top->next = element;

element->next = tmp;

\_length++;

return;

}

}

list\_node\_chain\* list\_chain::getByNumber(int position)

{

if (position < 0 || position >= \_length)

return nullptr;

else {

int i = 0;

list\_node\_chain\* top = head;

while (i != position && top != nullptr)

{

top = top->next;

i++;

}

return top;

}

}

void list\_chain::swap(list\_node\_chain\* left, list\_node\_chain\* right)

{

if (left == right)

return;

auto\* preLeft = head, \* preRight = head, \*top = head;

while (top != left) {

preLeft = top;

top = top->next;

}

top = head;

while (top!=right)

{

preRight = top;

top = top->next;

}

top = head;

if (preLeft == left) {

if (left->next == right) {

auto tmp = right->next;

left->next = tmp;

right->next = left;

this->head = right;

}

else {

this->head = right;

auto rN = right->next;

right->next = left->next;

left->next = rN;

preRight->next = left;

}

return;

}

else {

if (left != preRight) {

preLeft->next = right;

preRight->next = left;

auto rN = right->next;

right->next = left->next;

left->next = rN;

}

else {

preLeft->next = right;

left->next = right->next;

right->next = left;

}

}

}

void list\_chain::remove(int position)

{

list\_node\_chain\* top = getByNumber(position);

if (top == nullptr)

return;

else

{

//get previous element

if (position == 0) {

head = head->next;

\_length--;

return;

}

else

{

list\_node\_chain\* top\_previous = getByNumber(position - 1);

top\_previous->next = top->next;

\_length--;

delete top;

return;

}

}

}

void list\_chain::sort()

{

for (int i(0); i < length(); i++) {

for (int j(1); j < length(); j++) {

if (getByNumber(j)->value < getByNumber(j - 1)->value) {

swap(getByNumber(j - 1), getByNumber(j));

}

}

}

}

void list\_chain::task1()

{

//повторяющиеся переместить в конец

int len = length();

for (int i(0); i < len - 1; i++) {

//ищем элемент в массиве. Если его больше 1-го, то все найденые

//закидываем в конец, i = 0, len -= count(найденые эл-ы)

bool flag = false;

for (int j = i + 1; j < len; j++) {

if (getByNumber(i)->value == getByNumber(j)->value) {

this->swap(getByNumber(j), getByNumber(len-1));

--len;

flag = true;

}

}

if (flag) {

this->swap(getByNumber(i), getByNumber(len - 1));

len--;

}

}

}

std::string list\_chain::to\_string()

{

list\_node\_chain \*top = head;

std::string result = "";

for(int i(0); i < length(); i++)

{

result += std::to\_string(top->value) + " -> ";

top = top->next;

}

return result + "NULL";

}

/////////////////////////////////////////////////////////////////////////////////////////

#include "list\_array.h"

#include <iostream>

list\_array::list\_array(std::vector<list\_node\_array> \_values)

{

this->\_array = \_values;

this->head = (\_array.data());

}

void list\_array::append(list\_node\_array\* element)

{

auto last = this->getByNumber(\_array.size() - 1);

if(last != nullptr)

last->next = \_array.size();

\_array.push\_back(\*element);

}

void list\_array::insert(list\_node\_array\* element, int position\_after)

{

if (position\_after >= \_array.size() || element == nullptr)

return;

\_array.push\_back(\*element);

auto last = this->getByNumber(position\_after);

if (last != nullptr) {

auto tmp = last->next;

last->next = \_array.size() - 1;

\_array.back().next = tmp;

}

}

list\_node\_array\* list\_array::getByNumber(int position)

{

if (position < 0 || position >= \_array.size())

return nullptr;

else {

int i = 0;

auto top = &(\_array.data()[headNumber]);

while (i != position && top != nullptr)

{

top = &(\_array.data()[(top->next)]);

i++;

}

return top;

}

}

void list\_array::swap(list\_node\_array\* left, list\_node\_array\* right)

{

if (left == right)

return;

auto head = &(this->\_array[headNumber]);

auto\* preLeft = head, \* preRight = head, \* top = head;

while (top != left) {

preLeft = top;

top = &(\_array[top->next]);

}

top = head;

while (top != right)

{

preRight = top;

top = &(\_array[top->next]);

}

top = head;

if (preLeft == left) {

if (&(\_array[left->next]) == right) {

auto tmp = right->next;

left->next = tmp;

right->next = indexOfPointer(left);

this->head = right;

this->headNumber = indexOfPointer(right);

}

else {

this->head = right;

this->headNumber = indexOfPointer(right);

auto rN = right->next;

right->next = left->next;

left->next = rN;

preRight->next = indexOfPointer(left);

}

return;

}

else {

if (left != preRight) {

preLeft->next = indexOfPointer( right);

preRight->next = indexOfPointer( left);

auto rN = right->next;

right->next = left->next;

left->next = rN;

}

else {

preLeft->next = this->indexOfPointer(right);

left->next = right->next;

right->next = this->indexOfPointer(left);

}

}

}

void list\_array::remove(int position)

{

\_array.erase(\_array.begin() + position);

//tckb next > position => next--

for (auto iter = \_array.begin(); iter != \_array.end(); iter++) {

if ((\*iter).next > position)

(\*iter).next--;

}

}

void list\_array::sort()

{

for (int i(0); i < length(); i++) {

for (int j(1); j < length(); j++) {

if (getByNumber(j)->value < getByNumber(j - 1)->value) {

swap(getByNumber(j - 1), getByNumber(j));

}

}

}

}

std::string list\_array::to\_string()

{

if (this->length() > 0) {

std::string result = "";

for (int i(0); i < this->length(); i++) {

auto element = this->getByNumber(i);

result += std::string("(") + std::to\_string(element->value) + std::string("; ") + std::to\_string(element->next) + std::string(") -> ");

}

result += std::string("NULL");

return result;

}

else

return std::string("NULL");

}

bool list\_array::checkConsitency()

{

try {

//начиаем с головы, если пройденых эл-в меньше / больше длины вектора, то возвращаем false

int realLen = 0;

//////////////////////////////////////////////////////////////////////////////////////////

auto top = \_array.data();

while (top->next != -1 && \_array.size() + 100 >= realLen)

{

realLen++;

top = &\_array[top->next];

}

if (top->next == -1)

return true;

else {

std::cout << "Cycle found in list!!!!!!!!\r\n";

return false;

}

}

catch (std::exception& ex) {

std::cout << ex.what() << std::endl;

return false;

}

return false;

}

int list\_array::indexOfPointer(list\_node\_array\* ptr)

{

int number = 0;

auto top = \_array.data();

while (top != ptr && number < \_array.size())

{

number++;

top = &\_array.data()[number];

}

if (top == ptr)

return number;

else

return -1;

}

void list\_array::task1()

{

//повторяющиеся переместить в конец

int len = length();

for (int i(0); i < len - 1; i++) {

//ищем элемент в массиве. Если его больше 1-го, то все найденые

//закидываем в конец, i = 0, len -= count(найденые эл-ы)

bool flag = false;

for (int j = i + 1; j < len; j++) {

if (getByNumber(i)->value == getByNumber(j)->value) {

this->swap(getByNumber(j), getByNumber(len - 1));

--len;

flag = true;

}

}

if (flag) {

this->swap(getByNumber(i), getByNumber(len - 1));

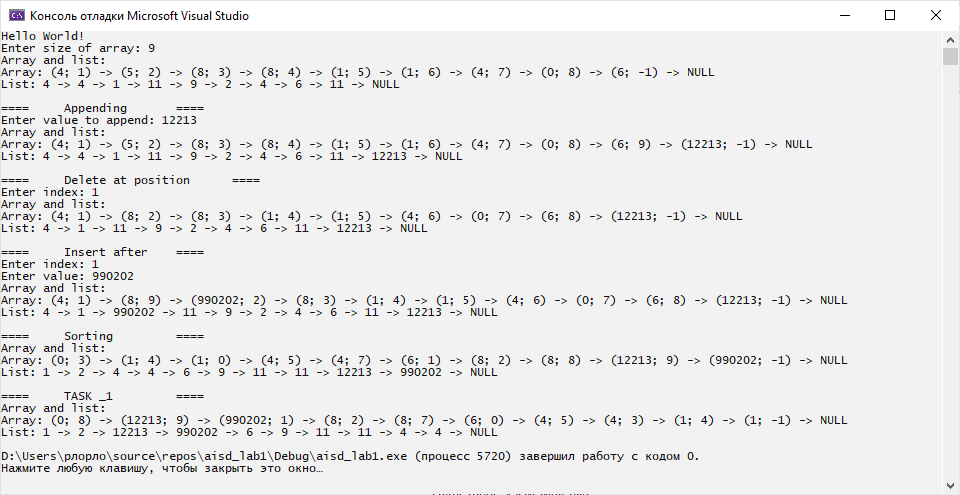
len--;

}

}

}

Результат выполнения:



**Вывод**: в ходе выполнения лабораторной работы изучил основы работы с линейными динамическими структурами (списки, очереди, стек, кольцевой буфер и т.д.)