1. Цель работы: исследование четырёх способов хранения множеств в памяти ЭВМ.

2. Задание на обработку множеств (формулировка из пособия).

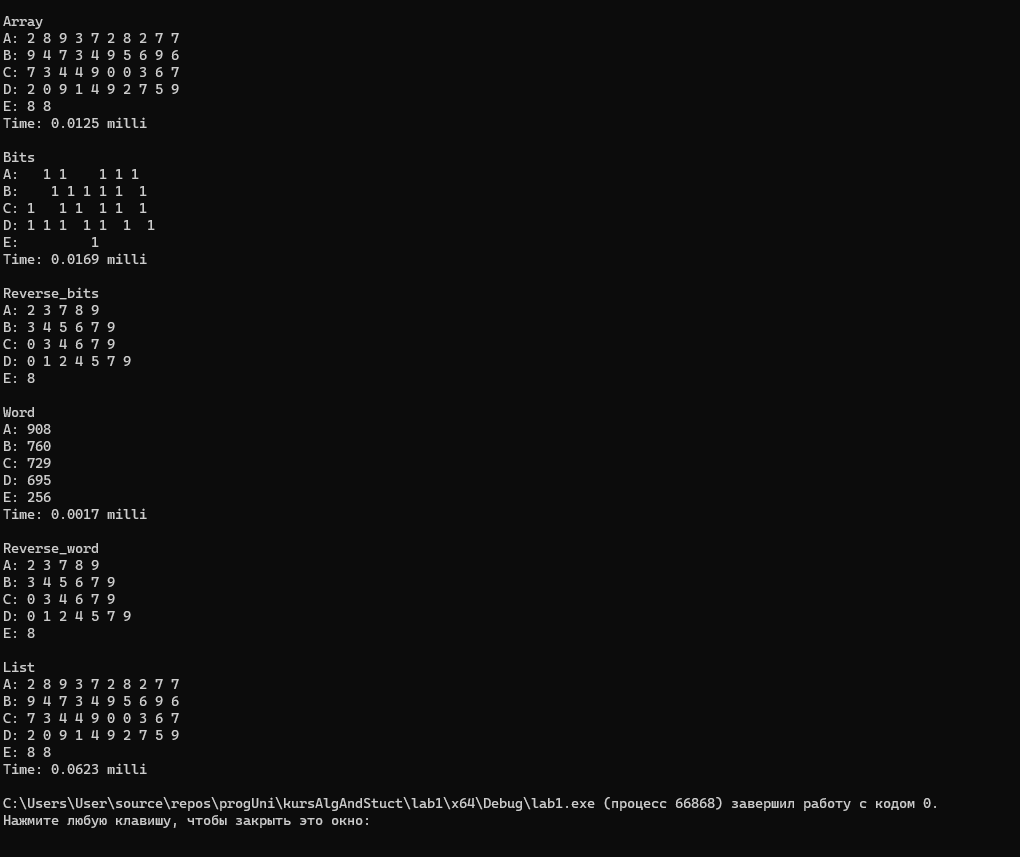
|  |  |  |
| --- | --- | --- |
| 17 | Десятичные цифры | Множество, содержащее цифры из *A*, не являющиеся общими для множеств *B* и *C*, и не встречающиеся в *D* |

3. Формализация задания: формула для вычисления пятого множества по четырём заданным.

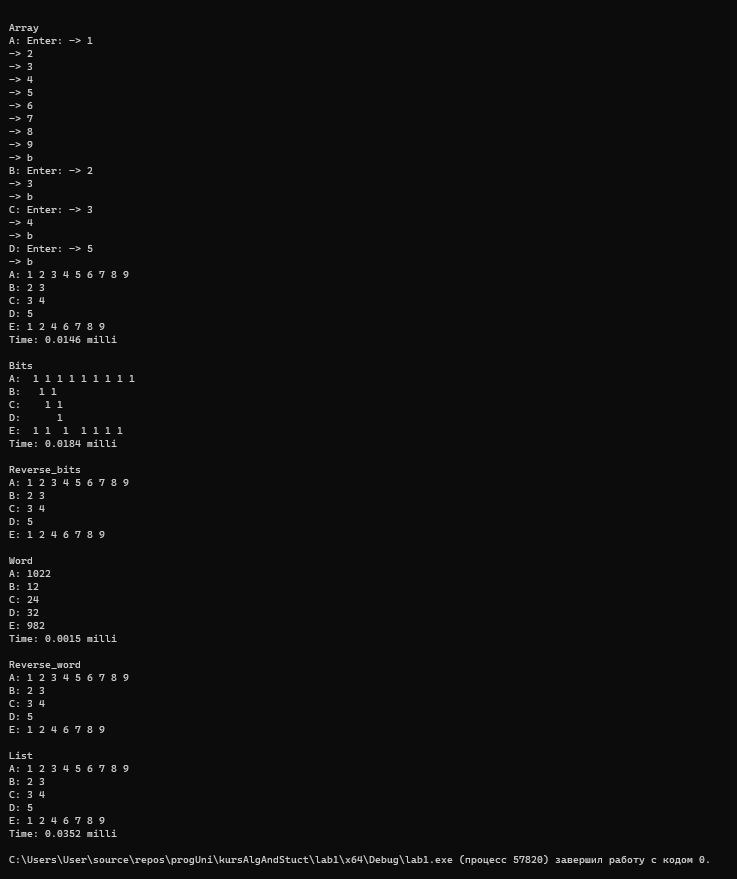
A/(D&B)/C

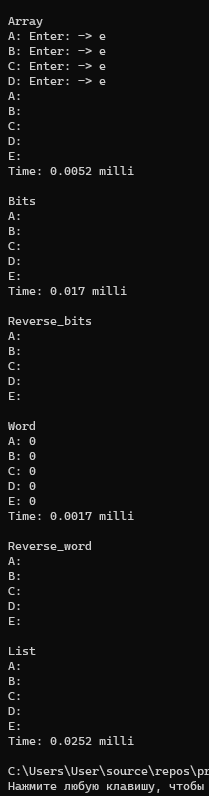
4. Контрольные тесты. Можно представить рисунки — скриншоты с результатами прогона тестов, при необходимости дополнив их пояснениями: где на рисунке исходные данные, где результат, что проверялось, какая структура данных использована и т. п. Рисунки не заменяют поясняющего текста, а прилагаются к нему.

A. Автоматическая генерация тестов



Б. Ручной ввод множеств





5. Временная сложность (ожидаемая и фактическая) для каждого из четырёх способов представления множеств.

|  |  |  |
| --- | --- | --- |
| Способ представления множеств | Ожидаемая | Фактическая |
| Массив | O(n2) | O(n2) |
| Список | O(n2) | O(n2) |
| Массив битов | O(n) | O(n) |
| Машинное слово | O(1) | O(1) |

6. Результаты измерения времени обработки каждым из способов, с пометкой, наблюдалась ли зависимость времени обработки от размера данных. Рекомендуется составить сводную таблицу.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Мощность множеств\* | 0 | 1 | 2 | 4 | 8 | 10 |
| Массив | 0.0063 | 0.0073 | 0.0075 | 0.0068 | 0.0069 | 0.0079 |
| Список | 0.0462 | 0.0546 | 0.0498 | 0.0475 | 0.0572 | 0.0446 |
| Массив битов | 0.0207 | 0.0157 | 0.0198 | 0.0168 | 0.0174 | 0.0187 |
| Машинное слово | 0.0015 | 0.0016 | 0.0019 | 0.0015 | 0.0014 | 0.0017 |

Количество времени, необходимое на обработку множества-массива и множества-списка, возрастает по мере увеличения его мощности, однако, явной квадратической зависимости не наблюдается.

Время, необходимое на обработку массива битов и машинного слова, стремится к константе.

7. Выводы о результатах испытания способов представления множеств в памяти и рекомендации по их применению в программах для ЭВМ. Необходимо указать достоинства, недостатки и возможную область применения для каждого из способов.

Проведено испытание четырех способов представления множеств в памяти (массив, линейный список, массив битов и машинное слово).

**Массив** дает относительно неплохие временные результаты в рамках данного задания, т.к. размер универсума (множество десятичных цифр) равен 10, а, следовательно, мощности множеств не превышают этого значения. Однако, предполагается, что при работе с более крупным размером универсума и более крупными множествами соответственно требуемое на их обработку время также существенно возрастет, что делает его хорошим выбором только для небольших значений |U|.

**Линейный список** показал наихудший временной результат среди всех представленных множеств. При этом видно, что даже при подаче «пустого» теста обработка множеств требует довольно длительного интервала времени (для сравнения: пустые списки обрабатываются дольше, нежели массивы, содержащие максимально возможное число элементов). Поэтому использование списков для небольших множеств кажется мне нецелесообразным.

К достоинствам как массива, так и линейного списка, относится то, что обрабатывается ровно столько элементов множества, сколько они содержат (в отличие от массива битов и машинного слова, подразумевающих перебор всего универсума вне зависимости от реальной мощности множеств) – однако, т.к. мощность универсума невелика, на практике это не дает ощутимых преимуществ, и потому оправдывает себя лишь при большом объеме данных.

**Массив битов** дает гораздо лучший временной результат, нежели обычный массив или список, т.к. временная сложность такого алгоритма составляет O(n), а у предыдущих двух - O(n2). С одной стороны, это позволяет сэкономить время работы алгоритма, однако, существенный недостаток заключается в том, что при работе с большим универсумом и фактически малых мощностях множеств, нам все равно приходится перебирать весь универсум, в то время как иначе можно было бы обойтись лишь просмотром реально существующих элементов.

Наконец, **машинное слово** показало наилучший временной результат, стремящийся к константе, что согласуется с предполагаемой временной сложностью алгоритма (O(1)), и потому его применение кажется мне наиболее оптимальной в данной задаче. Чрезвычайно ценным его преимуществом является и то, что машинное слово позволяет обработать любое количество множеств в одно выражение. Недостаток тот же: при большом размере универсума в большинстве случаев придется просматривать гораздо большее число элементов, нежели фактически содержится в множествах.

8. Список использованных источников.

1) Поздняков С. Н., Рыбин С. В. Дискретная математика: учебник для вузов. М.: Академия, 2008. – 448 с.

2) Хагерти Р. Дискретная математика для программистов. Изд. 2‑е, испр. — М.: Техносфера, 2012. — 400 с.;

3) Новиков Ф. А. Дискретная математика для программистов. — СПб.: Питер, 2000. — 304 с., ил.

4) Страуструп Б. Язык программирования С++. 2‑е доп. изд. — М.: Бином-пресс, 2001. – 1098 с.;

9. Приложение: исходные тексты всех программ

#include <iostream>

#include <vector>

#include <list>

#include <iterator>

#include <random>

#include <time.h>

#include <thread>

#include <chrono>

//func f()

char anconverting\_char(char var) {

var += '0';

return var;

}

char converting\_char(char var) {

var -= '0';

return var;

}

// rundom number

template <typename T>

T rundom\_set() {

T rundom\_set;

rundom\_set.reserve(10);

std::random\_device rd1;

std::mt19937 gen1{rd1()};

std::uniform\_int\_distribution<> dis1(0, 9);

int size = dis1(gen1);

for (std::vector<char>::size\_type i = 0; i < size; ++i) {

rundom\_set.emplace\_back(anconverting\_char(static\_cast<char>(dis1(gen1))));

}

return rundom\_set;

}

// array

std::vector<char> creat\_b\_conjunction\_c\_vec(

const std::vector<char>& b, const std::vector<char>& c)

{

std::vector<char> var;

for (const auto& i : b) {

for (const auto& k : c) {

if (i == k)

var.emplace\_back(i);

}

}

return var;

}

void difference\_sets\_a\_cb\_vec(std::vector<char>& var, const std::vector<char>& other) {

std::vector<char> vec{other};

for (auto&& i : vec) {

for (const auto& k : var) {

if (i == k)

i = '\*';

}

}

var = std::move(vec);

}

void difference\_sets\_a\_d\_vec(std::vector<char>& var, const std::vector<char>& other) {

for (auto&& i : var) {

for (const auto& k : other) {

if (i == k)

i = '\*';

}

}

}

std::vector<char> creat\_e\_vec(

const std::vector<char>& a, const std::vector<char>& b,

const std::vector<char>& c, const std::vector<char>& d)

{

std::vector<char> var = creat\_b\_conjunction\_c\_vec(b, c);

difference\_sets\_a\_cb\_vec(var, a);

difference\_sets\_a\_d\_vec(var, d);

std::vector<char> vec;

vec.reserve(var.size());

for (auto& i: var) {

if (i != '\*')

vec.push\_back(i);

}

return vec;

}

//bits

std::vector<char> anconvering\_vec\_bits(const std::vector<char> var, const int size) {

std::vector<char> other(var.size(), 0);

for (std::vector<char>::size\_type i = 0, k = 0; i < size; ++i) {

if (var[i]) {

char a = anconverting\_char(i);

other[k++] = a;

}

}

return other;

}

std::vector<char> converting\_vec\_bits(const std::vector<char> var) {

std::vector<char> bit\_var(10, 0);

for (std::vector<char>::size\_type i = 0; i < var.size(); ++i) {

bit\_var[converting\_char(var[i])] = 1;

}

return bit\_var;

}

std::vector<char> creat\_b\_bit\_conjunction\_c\_bit(

const std::vector<char> &b, const std::vector<char> &c)

{

std::vector<char> var(b.size(), 0);

for (std::vector<char>::size\_type i = 1; i < b.size(); ++i) {

if (b[i] && c[i])

var[i] = 1;

}

return var;

}

std::vector<char> difference\_sets\_bit(const std::vector<char>& var, const std::vector<char>& other) {

std::vector<char> vec\_a(var.size(), 0);

for (std::vector<char>::size\_type i = 0; i < var.size(); ++i) {

if (other[i] && !var[i])

vec\_a[i] = 1;

}

return vec\_a;

}

std::vector<char> creat\_e\_vec\_bit(

const std::vector<char>& a, const std::vector<char>& b,

const std::vector<char>& c, const std::vector<char>& d)

{

std::vector<char> var = creat\_b\_bit\_conjunction\_c\_bit(b, c);

var = difference\_sets\_bit(var, a);

var = difference\_sets\_bit(d, var);

return var;

}

//words

short int converting\_vec\_word\_machine(const std::vector<char>& var) {

short int w\_var = 0;

for (std::vector<char>::size\_type i = 0; i < var.size(); ++i) {

w\_var |= (1 << (static\_cast<int>(converting\_char(var[i]))));

}

return w\_var;

}

std::vector<char> anconverting\_vec\_word\_machine(const short int var,const int size) {

std::vector<char> other;

for (std::vector<char>::size\_type i = 0; i <= size; i++) {

if ((var >> i) & 1)

other.push\_back(anconverting\_char(i));

}

return other;

}

short int creat\_e\_word( short int a, const short int& b,

const short int c, const short int& d)

{

short int e = { a & ~(b & c) & ~d };

return e;

}

//list

std::list<char> creat\_list(const std::vector<char>& a) {

std::list<char> var;

for (const auto& i : a) {

var.push\_back(i);

}

return var;

}

std::list<char> creat\_b\_conjunction\_c\_list(const std::list<char>& b, const std::list<char>& c) {

std::list<char> var;

for (const auto& i : b) {

for (const auto& k : c) {

if (i == k)

var.push\_back(i);

}

}

return var;

}

void difference\_sets\_a\_cb\_list(std::list<char>& var, const std::list<char>& other) {

std::list<char> ls {other};

for (auto&& i : ls) {

for (const auto& k : var) {

if (i == k)

i = '\*';

}

}

var = std::move(ls);

}

void difference\_sets\_a\_d\_list(std::list<char>& var, const std::list<char>& other) {

for (auto&& i : var) {

for (const auto& k : other) {

if (i == k)

i = '\*';

}

}

}

std::list<char> creat\_e\_list(const std::list<char>& a, const std::list<char>& b,

const std::list<char>& c, const std::list<char>& d)

{

std::list<char> var = creat\_b\_conjunction\_c\_list(b, c);

difference\_sets\_a\_cb\_list(var, a);

difference\_sets\_a\_d\_list(var, d);

std::list<char> ls;

for (auto& i : var) {

if (i != '\*')

ls.push\_back(i);

}

return ls;

}

//out

template <typename T>

void out(const T& a, const char& name) {

std::this\_thread::sleep\_for(std::chrono::milliseconds(100));

std::cout << name << ": ";

for (const auto& i : a) {

if (i == '\x1')

std::cout << anconverting\_char(i) << ' ';

else

std::cout << i << ' ';

}

std::cout << std::endl;

}

template <typename T>

void out\_vecs\_time(const T& a, const T& b,

const T& c, T& d,

const T& e, const std::chrono::duration<double, std::milli> &res\_time)

{

out(a, 'A');

out(b, 'B');

out(c, 'C');

out(d, 'D');

out(e, 'E');

std::cout << "Time: " << res\_time.count() << " milli" << std::endl;

}

void out\_vecs(const std::vector<char>& a, const std::vector<char>& b,

const std::vector<char>& c, const std::vector<char>& d,

const std::vector<char>& e)

{

out(a, 'A');

out(b, 'B');

out(c, 'C');

out(d, 'D');

out(e, 'E');

}

void out\_word(short int a, short int b, short int c, short int d, short int e,

const std::chrono::duration<double, std::milli> &res\_time)

{

std::this\_thread::sleep\_for(std::chrono::milliseconds(100));

std::cout << "A: " << a << std::endl <<

"B: " << b << std::endl <<

"C: " << c << std::endl <<

"D: " << d << std::endl <<

"E: " << e << std::endl;

std::cout << "Time: " << res\_time.count() << " milli" << std::endl;

}

//interface

std::vector<char> in\_vec() {

std::vector<char> var;

int size = 10;

char ch = 0;

std::cout << "Enter: ";

for (std::vector<char>::size\_type i = 0; i < size; ++i) {

std::cout << "-> ";

std::cin >> ch;

if (ch == 'b')

break;

if (ch == 'e') {

var.reserve(10);

break;

}

var.push\_back(ch);

}

return var;

}

void interface\_vec(std::vector<char>& a, std::vector<char>& b,

std::vector<char>& c, std::vector<char>& d, int choice)

{

if (choice == 1) {

// standart

a = { '1','2','3','4','5','6','7','8','9' };

b = { '1','2' };

c = { '2','3' };

d = { '3' };

}else if(choice == 2){

// random

a = rundom\_set<std::vector<char>>();

b = rundom\_set<std::vector<char>>();

c = rundom\_set<std::vector<char>>();

d = rundom\_set<std::vector<char>>();

}

else if (choice == 3) {

std::cout << "A: ";

a = in\_vec();

std::cout << "B: ";

b = in\_vec();

std::cout << "C: ";

c = in\_vec();

std::cout << "D: ";

d = in\_vec();

}

}

// Вариант 17, E = A - (B \* C) - D

int main()

{

int size = 10;

// array

std::cout << std::endl << "Array" << std::endl;

std::vector<char> A;

std::vector<char> B;

std::vector<char> C;

std::vector<char> D;

// choice 1 - standart; 2 - rand; 3 - cin;

interface\_vec(A, B, C, D, 2);

auto time\_start1 = std::chrono::high\_resolution\_clock::now();

std::vector<char> E = creat\_e\_vec(A,B,C,D);

auto time\_end1 = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double, std::milli> res\_time1 = time\_end1 - time\_start1;

out\_vecs\_time(A, B, C, D, E, res\_time1);

//bits

std::cout << std::endl << "Bits" << std::endl;

auto time\_start2 = std::chrono::high\_resolution\_clock::now();

std::vector<char> bit\_a = converting\_vec\_bits(A);

std::vector<char> bit\_b = converting\_vec\_bits(B);

std::vector<char> bit\_c = converting\_vec\_bits(C);

std::vector<char> bit\_d = converting\_vec\_bits(D);

std::vector<char> bit\_e = creat\_e\_vec\_bit(bit\_a, bit\_b, bit\_c, bit\_d);

std::vector<char> reverse\_bit\_E = anconvering\_vec\_bits(bit\_e, size);

auto time\_end2 = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double, std::milli> res\_time2 = time\_end2 - time\_start2;

out\_vecs\_time(bit\_a, bit\_b, bit\_c, bit\_d, bit\_e, res\_time2);

std::cout << std::endl << "Reverse\_bits" << std::endl;

std::vector<char> reverse\_bit\_A = anconvering\_vec\_bits(bit\_a, size);

std::vector<char> reverse\_bit\_B = anconvering\_vec\_bits(bit\_b, size);

std::vector<char> reverse\_bit\_C = anconvering\_vec\_bits(bit\_c, size);

std::vector<char> reverse\_bit\_D = anconvering\_vec\_bits(bit\_d, size);

out\_vecs(reverse\_bit\_A, reverse\_bit\_B, reverse\_bit\_C, reverse\_bit\_D, reverse\_bit\_E);

//words

std::cout << std::endl << "Word" << std::endl;

auto time\_start3 = std::chrono::high\_resolution\_clock::now();

short int word\_a = converting\_vec\_word\_machine(A);

short int word\_b = converting\_vec\_word\_machine(B);

short int word\_c = converting\_vec\_word\_machine(C);

short int word\_d = converting\_vec\_word\_machine(D);

short int word\_z = ~word\_c;

short int word\_e = creat\_e\_word(word\_a, word\_b, word\_c, word\_d);

auto time\_end3 = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double, std::milli> res\_time3 = time\_end3 - time\_start3;

out\_word(word\_a, word\_b, word\_c, word\_d, word\_e, res\_time3);

std::cout << std::endl << "Reverse\_word" << std::endl;

std::vector<char> reverse\_word\_A = anconverting\_vec\_word\_machine(word\_a, size);

std::vector<char> reverse\_word\_B = anconverting\_vec\_word\_machine(word\_b, size);

std::vector<char> reverse\_word\_C = anconverting\_vec\_word\_machine(word\_c, size);

std::vector<char> reverse\_word\_D = anconverting\_vec\_word\_machine(word\_d, size);

std::vector<char> reverse\_word\_E = anconverting\_vec\_word\_machine(word\_e, size);

out\_vecs(reverse\_word\_A, reverse\_word\_B, reverse\_word\_C, reverse\_word\_D, reverse\_word\_E);

//list

std::cout << std::endl << "List" << std::endl;

auto time\_start4 = std::chrono::high\_resolution\_clock::now();

std::list<char> list\_a = creat\_list(A);

std::list<char> list\_b = creat\_list(B);

std::list<char> list\_c = creat\_list(C);

std::list<char> list\_d = creat\_list(D);

std::list<char> list\_e = creat\_e\_list(list\_a, list\_b, list\_c, list\_d);

auto time\_end4 = std::chrono::high\_resolution\_clock::now();

std::chrono::duration<double, std::milli> res\_time4 = time\_end4 - time\_start4;

out\_vecs\_time(list\_a, list\_b, list\_c, list\_d, list\_e, res\_time4);

}