Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский государственный электротехнический университет «ЛЭТИ»

им. В.И. Ульянова (Ленина)»

Кафедра ВТ

Отчет по лабораторной работе №1

Тема: «МНОЖЕСТВА»

Выполнили:

Зуб Виталий, 8307

Никулин Леонид, 8307

Преподаватель:

П. Г. Колинько

Тема «Множества»

Вариант 8

**Цель работы:** сравнительные испытания способов хранения множеств в памяти ЭВМ.

**Задание на обработку множества:**

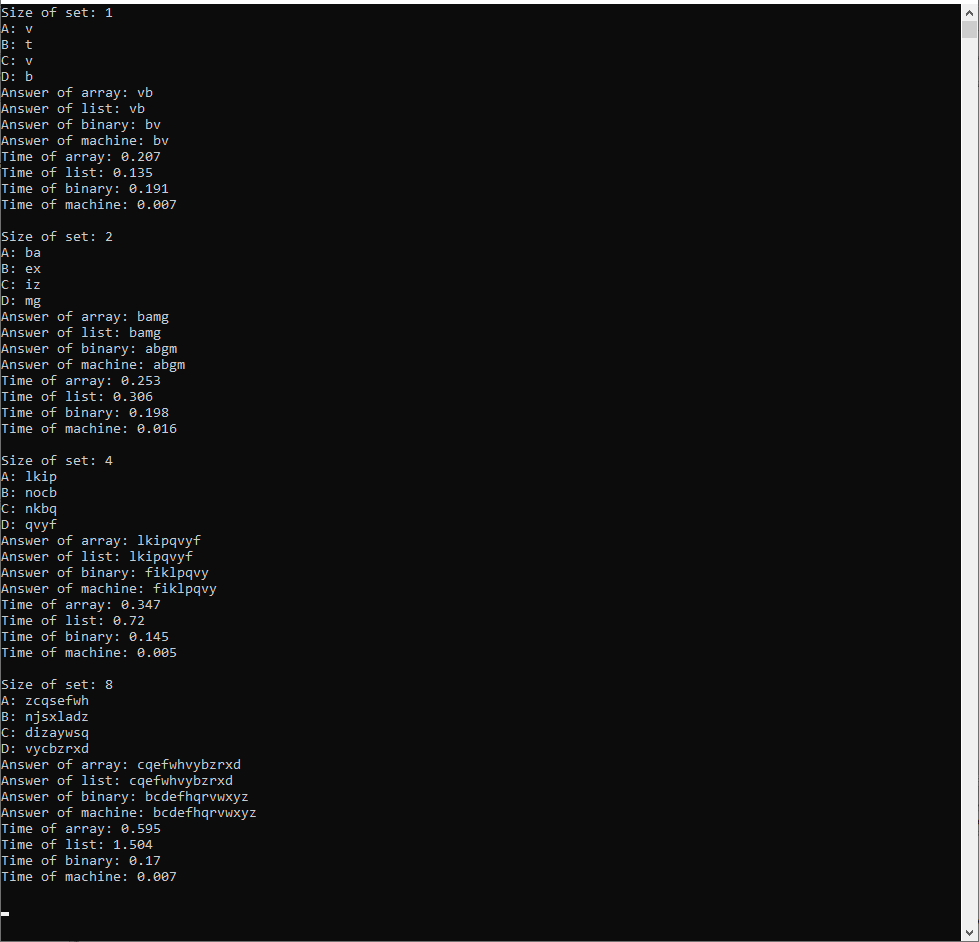
*Универсум:*Латинские буквы

*Что надо вычислить***:** Множество, содержащее буквы, имеющиеся во множестве A, но не являющиеся общими для B и C, и все буквы из D

**Формула:**

E = A / (B∩C) ∪ D

**Контрольные тесты:**

****

**Результаты измерений:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Кол-во элементов в исходных множествах | | | |
| Реализация множества | 1 | 2 | 4 | 8 |
| Массив | 0,207 | 0,253 | 0,347 | 0,595 |
| Список | 0,135 | 0,306 | 0,72 | 1,504 |
| Вектор битов | 0,191 | 0,198 | 0,145 | 0,17 |
| Машинное слово | 0,007 | 0,016 | 0,005 | 0,007 |

(Время рассчитывалось на основании проведения 1000000 повторений)

**Временная сложность:**

|  |  |  |
| --- | --- | --- |
| Реализация множества | Ожидаемая временная сложность | Фактическая временная сложность |
| Массив | O(n2) | O(n) |
| Список | O(n2) | O(n2) |
| Массив битов | O(1) | O(1) |
| Машинное слово | O(1) | O(1) |

**Выводы:**

**Машинное слово:**

Наиболее эффективный способ представления множеств в памяти – это *машинное слово*. Время исполнения алгоритма не зависело от размера множества O(1) и происходило всегда быстрее других реализаций. Множество рассматривается как подмножество (S) универсума (U), представленное в виде характеристического вектора из n элементов, i-й элемент в этом векторе равен 1 тогда и только тогда, когда i-й элемент множества U принадлежит S, иначе он равен 0.

**Список:**

Наименее эффективный способ представления множеств в памяти – это *список*. Время исполнения алгоритма зависело от размера множества O(n2) и происходило всегда дольше других реализаций. Такие результаты объясняются необходимостью работать с указателями, что значительно увеличивает время обработки. В этом случае выделяется ровно столько памяти, сколько необходимо. Перенос элемента из одного множества в другое не требует дополнительной памяти. Однако, данный способ имеет недостатки:

· необходимость хранить указатель на следующий элемент;

· минимальная область, выделяемая в динамической памяти, часто больше, чем требуется для хранения одного элемента множества с указателем.

**Вектор битов:**

Реализация при помощи *вектора битов* также, как и машинное слово, имеет временную сложность O(1), данный способ представления множества в памяти удобен тем, что можно определять принадлежность i-го элемента множеству за время не зависящее от его размера. Основные операции над элементами множества можно осуществлять как логические операции над соответствующими битами (||, &&, !). Недостаток данного способа состоит в том, что операции объединения и пересечения множеств занимают время пропорциональное мощности универсума, а не мощности рассматриваемых множеств.

**Массив:**

Тестирование реализации алгоритма с использованием *массива* показало линейную временную сложность, данный способ представления множества в памяти является экономным и удобен лишь тогда, когда количество элементов множества известно и элементы – данные одного типа. Если множество создается в результате вычислений, то его размерность не всегда известна заранее, поэтому приходится выделять максимально возможное для данного вычисления количество памяти, либо размерность универсума, что нерационально. Также память, выделенная под множество, может быть переполнена, что потребует дополнительного выделения памяти большего размера и переноса туда элементов, что является дорогой операцией, либо невыполнимой, если в памяти нет непрерывного участка данного размера.

**Список использованной литературы:**

1. Колинько П.Г. Алгоритмы и структуры данных. Часть 1: Методические указания к самостоятельной работе на ПЭВМ и курсовому проектированию. Вып. 1909. – СПб., 2019. – 71с
2. clock – C++. cplusplus.com/reference/ctime/clock/

**Приложение: Текст программы**

#include <iostream>

#include <conio.h>

#include <ctime>

using namespace std;

const int count = 4;

const int sizeUniversum = 26;

struct Set {

char c = '\0';

Set \*next = NULL;

};

bool checkSetList(Set \*set, char c) {

for (Set \*p = set; p; p = p->next) {

if (p->c == c)

return true;

}

return false;

}

bool checkSetArray(const char \*set, int size, char c) {

for (int i = 0; i < size; i++) {

if (set[i] == c)

return true;

}

return false;

}

void addList(Set \*set, char c) {

if (checkSetList(set, c)) return;

for (Set \*p = set; p; p = p->next) {

if (p->c == '\0') {

p->c = c;

} else if (p->next == nullptr) {

Set \*current = new Set;

current->c = c;

p->next = current;

break;

}

}

}

char \*\*getRandomSet(int sizeCount) {

char \*\*collections = new char \*[count];

for (int i = 0; i < count; i++) {

collections[i] = new char[sizeCount];

for (int j = 0; j < sizeCount; j++) {

char c;

do {

c = 'a' + (rand() % sizeUniversum);

} while (checkSetArray(collections[i], sizeCount, c));

collections[i][j] = c;

}

}

return collections;

}

void printSets(char \*\*collections, int size) {

cout << "Size of set: " << size << endl;

char set = 'A';

for (int i = 0; i < count; i++) {

cout << set << ": ";

for (int j = 0; j < size; ++j) {

cout << collections[i][j];

}

cout << endl;

set++;

}

}

void printList(Set \*set) {

for (Set \*p = set; p; p = p->next) {

cout << p->c;

}

}

Set \*convertToList(char \*\*collections, int size) {

Set \*listCollections = new Set[count];

for (int i = 0; i < count; ++i) {

for (int j = 0; j < size; ++j) {

addList(&listCollections[i], collections[i][j]);

}

}

return listCollections;

}

bool \*\*convertToBinary(char \*universum, char \*\*collections, int size) {

bool \*\*binaryCollections = new bool \*[count];

for (int i = 0; i < count; i++) {

binaryCollections[i] = new bool[sizeUniversum];

for (int j = 0; j < sizeUniversum; ++j) {

binaryCollections[i][j] = false;

}

}

for (int i = 0; i < count; ++i) {

for (int j = 0; j < size; ++j) {

int k;

for (k = 0; collections[i][j] != universum[k]; ++k) {}

binaryCollections[i][k] = true;

}

}

return binaryCollections;

}

long \*convertToMachine(char \*universum, char \*\*collections, int size) {

long \*machineCollections = new long[count];

for (int i = 0; i < count; i++) {

machineCollections[i] = 0;

}

for (int i = 0; i < count; ++i) {

for (int j = 0; j < size; ++j) {

int k;

for (k = 0; collections[i][j] != universum[k]; ++k) {}

machineCollections[i] ^= (1 << k);

}

}

return machineCollections;

}

double getArrayTime(char \*\*collections, int size, bool writeAnswer) {

auto start = clock();

int k = 0;

char \*answer = new char[count \* size];

// Use case

for (int i = 0; i < size; i++) {

if (!(checkSetArray(collections[1], size, collections[0][i]) &&

checkSetArray(collections[2], size, collections[0][i]))) {

if (!checkSetArray(answer, k, collections[0][i])) {

answer[k] = collections[0][i];

k++;

}

}

}

for (int i = 0; i < size; i++) {

if (!checkSetArray(answer, k, collections[3][i])) {

answer[k] = collections[3][i];

k++;

}

}

if (writeAnswer) {

cout << "Answer of array: ";

for (int i = 0; i < k; i++) {

cout << answer[i];

}

cout << endl;

}

auto stop = clock();

return (stop - start) / (double) CLOCKS\_PER\_SEC;

}

double getListTime(Set \*collections, bool writeAnswer) {

auto start = clock();

Set answer;

for (Set \*p = &collections[0]; p; p = p->next) {

if (!(checkSetList(&collections[1], p->c) && checkSetList(&collections[2], p->c))) {

if (!checkSetList(&answer, p->c)) {

addList(&answer, p->c);

}

}

}

for (Set \*p = &collections[3]; p; p = p->next) {

if (!checkSetList(&answer, p->c)) {

addList(&answer, p->c);

}

}

if (writeAnswer) {

cout << "Answer of list: ";

printList(&answer);

cout << endl;

}

auto stop = clock();

return (stop - start) / (double) CLOCKS\_PER\_SEC;

}

double getBinaryTime(char \*universum, bool \*\*collections, bool writeAnswer) {

auto start = clock();

bool answer[sizeUniversum];

for (int i = 0; i < sizeUniversum; i++) {

answer[i] = false;

answer[i] |= collections[0][i] & !(collections[1][i] & collections[2][i]);

}

for (int i = 0; i < sizeUniversum; i++) {

answer[i] |= collections[3][i];

}

if (writeAnswer) {

cout << "Answer of binary: ";

for (int i = 0; i < sizeUniversum; i++) {

if (answer[i]) {

cout << universum[i];

}

}

cout << endl;

}

auto stop = clock();

return (stop - start) / (double) CLOCKS\_PER\_SEC;

}

double getMachineTime(char \*universum, long \*collections, bool writeAnswer) {

auto start = clock();

long answer = 0;

answer |= collections[0] & ~(collections[1] & collections[2]);

answer |= collections[3];

if (writeAnswer) {

cout << "Answer of machine: ";

for (int i = 0; i < sizeUniversum; i++) {

if ((answer >> i & 1)) {

cout << universum[i];

}

}

cout << endl;

}

auto stop = clock();

return (stop - start) / (double) CLOCKS\_PER\_SEC;

}

int main() {

srand(time(nullptr));

const int testCount = 1000000;

int setSizeCount[] = {1, 2, 4, 8};

char universum[sizeUniversum];

for (int i = 0, j = 'a'; j <= 'z'; ++i, ++j) {

universum[i] = j;

}

for (auto i : setSizeCount) {

double timeArray = 0;

double timeList = 0;

double timeBinary = 0;

double timeMachine = 0;

char \*\*collections = getRandomSet(i);

Set \*listCollections = convertToList(collections, i);

bool \*\*binaryCollections = convertToBinary(universum, collections, i);

long \*machineCollections = convertToMachine(universum, collections, i);

printSets(collections, i);

for (int j = 0; j < testCount; j++) {

timeArray += getArrayTime(collections, i, j == 0);

timeList += getListTime(listCollections, j == 0);

timeBinary += getBinaryTime(universum, binaryCollections, j == 0);

timeMachine += getMachineTime(universum, machineCollections, j == 0);

}

//timeArray /= testCount;

//timeList /= testCount;

//timeBinary /= testCount;

//timeMachine /= testCount;

cout << "Time of array: " << timeArray << endl;

cout << "Time of list: " << timeList << endl;

cout << "Time of binary: " << timeBinary << endl;

cout << "Time of machine: " << timeMachine << endl << endl;

}

\_getch();

return 0;

}