МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ «ЛЭТИ» ИМ.В.И.УЛЬЯНОВА (ЛЕНИНА)»

ФАКУЛЬТЕТ КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ИНФОРМАТИКИ
КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Зачётная работа №1
по дисциплине «Алгоритмы и структуры данных»
на тему «Множества в памяти ЭВМ»

Выполнили студенты группы 3312

Лебедев И.А.

Шарапов И.Д.

Принял старший преподаватель Колинько П.Г.

Санкт-Петербург
2024

Содержание

Цель работы

Сравнительное исследование четырёх способов хранения множеств в памяти ЭВМ.

Описание задания

Множество десятичных цифр, содержащее цифры, общие для множеств A и B, а также все цифры из множеств C и D.

Формула для вычисления пятого множества

$$E = A \wedge B \vee C \vee D$$

Контрольные тесты

```
Test 3:
4 8 3
4 9 1
8 9 4
9 1 0
Array result: 4 8 9 1 0 in 1100 nanoseconds
List result:  4 8 9 1 0 in 800 nanoseconds
Bits result:  0 1 4 8 9 in 300 nanoseconds
MWord result: 0 1 4 8 9 in 100 nanoseconds
```

Рис. 1

```
Test 5:
0 7 4 1 2
1 4 2 8 7
5 8 9 4 6
6 8 3 9 0
Array result: 7 4 1 2 5 8 9 6 3 0 in 1300 nanoseconds
List result:  7 4 1 2 5 8 9 6 3 0 in 1500 nanoseconds
Bits result:  0 1 2 3 4 5 6 7 8 9 in 300 nanoseconds
MWord result: 0 1 2 3 4 5 6 7 8 9 in 100 nanoseconds
```

Рис. 2

```
Test 11:
0 2
0 3 4
3 5
3 4
Array result: 0 3 5 4 in 1000 nanoseconds
List result:  0 3 5 4 in 7700 nanoseconds
Bits result:  0 3 4 5 in 300 nanoseconds
MWord result: 0 3 4 5 in 100 nanoseconds
```

Рис. 3

Временная сложность

Таблица 1. Способы представления и временная сложность обработки

Способ представления	Временная сложность	
	Ожидаемая	Фактическая
Массив символов	$O(n^2)$	$O(n^2)$
Список		$O(n^2)$
Универсум	$O(\|U\|)$	$O(1)$
Машинное слово		$O(1)$

Пояснения: Для множества, представленного как массив символов или список, выполнение двуместной операции требует проверки всех возможных пар элементов, что для множества размера n даёт сложность $O(n^2)$. Если вычисление выражения сводится к последовательности таких операций, общая сложность будет аналогичной.

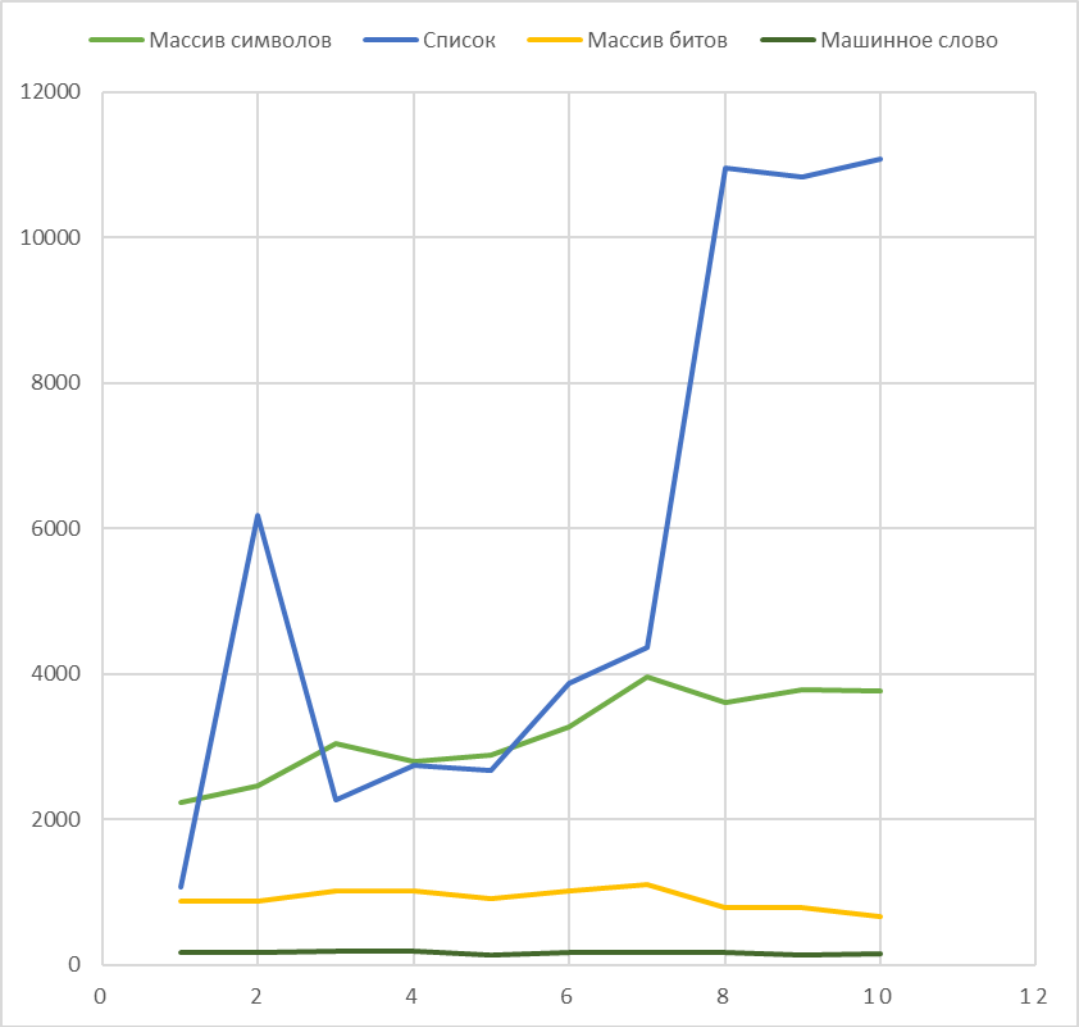
Для множеств, представленных как отображение на универсум, ожидаемое количество шагов для выполнения каждой двуместной операции соответствует мощности универсума. Поскольку вычисление выражения выполняется как последовательность двуместных операций, каждая из которых реализуется циклом, проходящим по всему множеству, фактическая временная сложность алгоритма совпадает с ожидаемой. Если мощность универсума фиксирована, сложность вычислений можно считать постоянной (константной).

Результат измерения времени обработки для каждого из способов

Таблица 2. Результаты измерения времени обработки множеств.

Мощность множеств	Количество *наносекунд* при обработке множеств при различных способах представления			
	Массив символов	Список	Массив битов	Машинное слово
1	*2228*	*1064*	*870*	*161*
2	*2457*	*6182*	*882*	*177*
3	*3048*	*2261*	*1017*	*183*
4	*2800*	*2743*	*1015*	*187*
5	*2891*	*2677*	*918*	*140*
6	*3271*	*3864*	*1011*	*163*
7	*3958*	*4358*	*1103*	*176*
8	*3612*	*10958*	*787*	*174*
9	*3782*	*10841*	*782*	*141*
10	*3786*	*11075*	*661*	*150*

Диаграмма 1. Оценка измерения времени обработки множеств.



Пояснения: Данные являются средним арифметическим полученным при запуске файла *AutoTest.cpp* на двух разных ПК. Данные занесли в табличку и построили диаграмму в программе MS Excel.

Выводы

В ходе данной работы было установлено, что машинное слово является наиболее эффективным способом представления множеств для быстрой обработки. Этот метод рекомендуется применять, когда существует простая логическая функция, позволяющая сопоставить элемент множества с соответствующим порядковым номером бита, и размер универсума не превышает разрядности слова.

Аналогично, использование массива битов также подходит для представления множеств, однако в этом случае мощность универсума должна оставаться ограниченной.

Наиболее медленным методом представления оказались списки. Данный способ стоит использовать, когда изначально неизвестна мощность создаваемого множества и нет возможности выделить память под все элементы сразу.

Представление множества в виде массива элементов следует применять, если можно с достаточной точностью определить размер массива, а мощность универсума слишком велика для использования вектора битов или машинного слова.

Список используемых источников

Колинько П. Г. Пользовательские структуры данных: Методические указания по дисциплине «Алгоритмы и структуры данных, часть 1». — СПб.: СПбГЭТУ «ЛЭТИ», 2024. — 64 с. (вып.2408).

Приложение

Код Array.cpp

```
#include <iostream>
#include <chrono>

#ifdef LOCAL
    bool loc = true;
#else
    bool loc = false;
#endif

void scan_set(char t[11]) {
    char x;
    int cnt = 0;
    do {
        x = (char) getc(stdin);
        if ('0' <= x && x <= '9') {
            bool found = false;
            for (int i = 0; i < cnt; ++i) {
                if (t[i] == x) {
                    found = true;
                    break;
                }
            }
            if (!found) {
                t[cnt++] = x;
            }
        }
    } while (x != '\n');
}

int main() {
    char a[11]{}, b[11]{}, c[11]{}, d[11]{}, e[11]{};
    int cnt_e = 0;

    // scan
    if (loc) printf("A: ");
    scan_set(a);
    if (loc) printf("B: ");
    scan_set(b);
    if (loc) printf("C: ");
    scan_set(c);
    if (loc) printf("D: ");
    scan_set(d);

    auto start = std::chrono::high_resolution_clock::now();
    // removing repetitions
    for (int i = 0; i < 11; ++i) {
        for (int j = i + 1; j < 11; ++j) {
            if (a[i] == a[j]) a[j] = '\000';
        }
    }

    // e = a & b
    for (char i: a) {
        for (char j: b) {
            if (i == j && i != '\000') {
                e[cnt_e++] = i;
            }
        }
    }

    // e |= c
    for (char i: c) {
        if (i != '\000') {
            bool found = false;
            for (int j = 0; j < cnt_e; ++j) {
                if (e[j] == i) {
                    found = true;
                    break;
                }
            }
            if (!found) {
                e[cnt_e++] = i;
            }
```

```
        }
    }

    // e |= d
    for (char i: d) {
        if (i != '\000') {
            bool found = false;
            for (int j = 0; j < cnt_e; ++j) {
                if (e[j] == i) {
                    found = true;
                    break;
                }
            }
            if (!found) {
                e[cnt_e++] = i;
            }
        }
    }
    auto stop = std::chrono::high_resolution_clock::now();

    // print
    if (loc) printf("E: ");
    for (int i = 0; i < cnt_e; ++i) {
        printf("%c ", e[i]);
    }
    printf("in %lli nanoseconds", std::chrono::duration_cast<std::chrono::nanoseconds>(stop - start).count());

    return 0;
}
```

Код List.cpp

```
#include <iostream>
#include <chrono>

#ifdef LOCAL
    bool loc = true;
#else
    bool loc = false;
#endif

struct Node {
    char data;
    Node *next;
};

Node *createNode(char data) {
    Node *newNode = new Node();
    newNode->data = data;
    newNode->next = nullptr;
    return newNode;
}

void insertNode(Node **head, char data) {
    Node *current = *head;
    while (current != nullptr) {
        if (current->data == data) {
            return; // Node already exists
        }
        current = current->next;
    }
    Node *newNode = createNode(data);
    if (*head == nullptr) {
        *head = newNode;
    } else {
        Node *lastNode = *head;
        while (lastNode->next != nullptr) {
            lastNode = lastNode->next;
        }
        lastNode->next = newNode;
    }
}

void scanSet(Node **head) {
    char x;
    do {
        x = (char) getc(stdin);
```

```
        if ('0' <= x && x <= '9') {
            insertNode(head, x);
        }
    } while (x != '\n');
}

void bitwiseAnd(Node *headA, Node *headB, Node **result) {
    Node *currentA = headA;
    while (currentA != nullptr) {
        Node *currentB = headB;
        bool found = false;
        while (currentB != nullptr) {
            if (currentA->data == currentB->data) {
                found = true;
                break;
            }
            currentB = currentB->next;
        }
        if (found) {
            insertNode(result, currentA->data);
        }
        currentA = currentA->next;
    }
}

void bitwiseOr(Node *headA, Node *headB, Node **result) {
    Node *currentA = headA;
    while (currentA != nullptr) {
        insertNode(result, currentA->data);
        currentA = currentA->next;
    }
    Node *currentB = headB;
    while (currentB != nullptr) {
        bool found = false;
        Node *currentResult = *result;
        while (currentResult != nullptr) {
            if (currentB->data == currentResult->data) {
                found = true;
                break;
            }
            currentResult = currentResult->next;
        }
        if (!found) {
            insertNode(result, currentB->data);
        }
        currentB = currentB->next;
    }
}

int main() {
    Node *headA = nullptr, *headB = nullptr, *headC = nullptr, *headD = nullptr, *headE = nullptr;

    // scan
    if (loc) printf("A: ");
    scanSet(&headA);
    if (loc) printf("B: ");
    scanSet(&headB);
    if (loc) printf("C: ");
    scanSet(&headC);
    if (loc) printf("D: ");
    scanSet(&headD);

    auto start = std::chrono::high_resolution_clock::now();
    // e = a & b
    bitwiseAnd(headA, headB, &headE);
    // e = c | e
    bitwiseOr(headC, headE, &headE);
    // e = d | e
    bitwiseOr(headD, headE, &headE);
    auto stop = std::chrono::high_resolution_clock::now();

    // print
    if (loc) printf("E: ");
    Node *current = headE;
    while (current != nullptr) {
        printf("%c ", current->data);
        current = current->next;
    }
```

```
    printf("in %lli nanoseconds", std::chrono::duration_cast<std::chrono::nanoseconds>(stop - start).count());

    return 0;
}
```

Код Bits.cpp

```
#include <iostream>
#include <chrono>

#ifdef LOCAL
    bool loc = true;
#else
    bool loc = false;
#endif

void scan_set(bool t[10]) {
    char x;
    do {
        x = (char) getc(stdin);
        if ('0' <= x && x <= '9') {
            t[x - '0'] = true;
        }
    } while (x != '\n');
}

int main() {
    bool a[10]{}, b[10]{}, c[10]{}, d[10]{}, e[10]{};

    // scan
    if (loc) printf("A: ");
    scan_set(a);
    if (loc) printf("B: ");
    scan_set(b);
    if (loc) printf("C: ");
    scan_set(c);
    if (loc) printf("D: ");
    scan_set(d);

    auto start = std::chrono::high_resolution_clock::now();
    // e = a & b
    for (int i = 0; i < 10; ++i) {
        e[i] = a[i] && b[i];
    }

    // e |= c
    for (int i = 0; i < 10; ++i) {
        e[i] = e[i] || c[i];
    }

    // e |= d
    for (int i = 0; i < 10; ++i) {
        e[i] = e[i] || d[i];
    }
    auto stop = std::chrono::high_resolution_clock::now();

    // print
    if (loc) printf("E: ");
    for (int i = 0; i < 10; ++i) {
        if (e[i]) printf("%i ", i);
    }
    printf("in %lli nanoseconds", std::chrono::duration_cast<std::chrono::nanoseconds>(stop - start).count());

    return 0;
}
```

Код MWord.cpp

```
#include <iostream>
#include <chrono>

#ifdef LOCAL
    bool loc = true;
#else
    bool loc = false;
#endif
```

```
unsigned short scan_set() {
    unsigned short t = 0;
    char x;
    do {
        x = (char) getc(stdin);
        if ('0' <= x && x <= '9') {
            t |= 1 << (int) (x - '0');
        }
    } while (x != '\n');
    return t;
}

int main() {
    unsigned short a = 0, b = 0, c = 0, d = 0, e;

    // scan
    if (loc) printf("A: ");
    a = scan_set();
    if (loc) printf("B: ");
    b = scan_set();
    if (loc) printf("C: ");
    c = scan_set();
    if (loc) printf("D: ");
    d = scan_set();

    auto start = std::chrono::high_resolution_clock::now();
    e = a & b | c | d;
    auto stop = std::chrono::high_resolution_clock::now();

    //print
    if (loc) printf("E: ");
    for (int i = 0; i < 10; ++i) {
        if ((e >> i) & 1) printf("%i ", i);
    }
    printf("in %lli nanoseconds", std::chrono::duration_cast<std::chrono::nanoseconds>(stop - start).count());

    return 0;
}
```

Код Generator.cpp

```
#include <iostream>
#include <algorithm>
#include <random>

#ifdef LOCAL
    bool loc = true;
#else
    bool loc = false;
#endif

void generator(char a[], int &len) {
    std::mt19937 rnd(std::random_device{}());
    len = std::uniform_int_distribution(1, 10)(rnd);

    char digits[10] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};

    for (int i = 0; i < len; ++i) {
        if (std::uniform_int_distribution<int>(0, 1)(rnd)) {
            a[i] = digits[i];
        } else {
            a[i] = '\0';
        }
    }

    std::shuffle(a, a + len, rnd);
}

void generatorWithFixedLen(char a[], int len) {
    std::mt19937 rnd(std::random_device{}());

    char digits[10] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};

    for (int i = 9; i > 0; --i) {
        int j = std::uniform_int_distribution<int>(0, i)(rnd);
        std::swap(digits[i], digits[j]);
```

```
    }

    for (int i = 0; i < len; ++i) {
        a[i] = digits[i];
    }
}

int main(int argc, char *argv[]) {
    int len = 0;
    if (loc) {
        std::cout << "Enter length of generated sets (from 0 to 10):\n";
        std::cin >> len;
    } else {
        if (argc > 1) {
            len = std::atoi(argv[1]);
        }
    }
    for (int i = 0; i < 4; ++i) {
        char a[10];
        if (!loc && argc <= 1) {
            generator(a, len);
        } else {
            generatorWithFixedLen(a, len);
        }

        for (int j = 0; j < len; ++j) {
            if (a[j] != '\0') {
                std::cout << a[j] << " ";
            }
        }
        std::cout << "\n";
    }
    return 0;
}
```

Код AutoTest.cpp

```
#include <cstdlib>
#include <chrono>
#include <iostream>
#include <fstream>

void out_print(const char name[]) {
    std::string line;
    std::ifstream file(name);
    getline(file, line);
    file.close();
    std::cout << line;
}

int main() {

    // compile
#ifdef LOCAL
    system("g++ ../Generator.cpp -o Generator");
    system("g++ ../Array.cpp -o Array");
    system("g++ ../List.cpp -o List");
    system("g++ ../Bits.cpp -o Bits");
    system("g++ ../MWord.cpp -o MWord");
#else
    system("g++ Generator.cpp -o Generator");
    system("g++ Array.cpp -o Array");
    system("g++ List.cpp -o List");
    system("g++ Bits.cpp -o Bits");
    system("g++ MWord.cpp -o MWord");
#endif

    // running
    for (int i = 1; i <= 11; ++i) {
        if (i < 11) system(std::format("Generator {} > input.txt", i).c_str());
        else system("Generator > input.txt");

        std::cout << "Test " << i << ":\n";
        std::ifstream file("input.txt");
        std::string line;
        while (std::getline(file, line)) {
            std::cout << line << "\n";
```

```
        }
        file.close();

        system("Array < input.txt > out_array.txt");
        std::cout << "Array result: ";
        out_print("out_array.txt");

        system("List < input.txt > out_list.txt");
        std::cout << "\nList result:  ";
        out_print("out_list.txt");

        system("Bits < input.txt > out_bits.txt");
        std::cout << "\nBits result:  ";
        out_print("out_bits.txt");

        system("MWord < input.txt > out_mword.txt");
        std::cout << "\nMWord result: ";
        out_print("out_mword.txt");
        std::cout << "\n\n";
    }

    return 0;
}
```