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, которые одновременно принадлежат множествам A и B, а также цифры из множеств C и D.

Формула для вычисления пятого множества

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

Контрольные тесты

Рассмотрим результаты программы над разными множествами. Каждый пример состоит из четырёх множеств (A, B, C, D), которые объединяются и пересекаются для формирования нового множества E. В качестве результата для каждого способа представления приводится итоговое множество E и время, затраченное на выполнение операций.

В первом примере (Рис. 1) пересечение множеств A и B даёт множество из одного элемента. После чего добавляются все элементы из C и D, так чтобы не было дубликатов.

```
Test 2:
A: 4 3
B: 5 3
C: 8 1
D: 1 6
Array result: 3 8 1 6 in 300 nanoseconds
List result:  6 8 1 3 in 7600 nanoseconds
Bits result:  1 3 6 8 in 300 nanoseconds
MWord result: 1 3 6 8 in 100 nanoseconds
```

Рисунок 1 – Пример с длиной множества 2

Во втором примере (Рис. 2) также можно наблюдать обработку дубликатов и корректное выполнение формулы.

```
Test 4:
A: 9 0 3 7
B: 6 8 5 9
C: 4 6 5 1
D: 8 7 5 0
Array result: 9 4 6 5 1 8 7 0 in 500 nanoseconds
List result:  8 7 0 4 6 5 1 9 in 700 nanoseconds
Bits result:  0 1 4 5 6 7 8 9 in 300 nanoseconds
MWord result: 0 1 4 5 6 7 8 9 in 100 nanoseconds
```

Рисунок 2 – Пример с длиной множества 4

В третьем примере (Рис. 3) демонстрируется работа с множествами разной произвольной длины.

```
Test 11:
A: 3 9 2
B: 0 3 4
C: 3 0
D: 5 4 6 2 0
Array result: 3 0 5 4 6 2 in 400 nanoseconds
List result:  5 4 6 2 0 3 in 500 nanoseconds
Bits result:  0 2 3 4 5 6 in 300 nanoseconds
MWord result: 0 2 3 4 5 6 in 100 nanoseconds
```

Рисунок 3 – Пример с произвольной длиной множества

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	$O(n^2)$	$O(n^2)$
Универсум	$O(\|U\|)$	$O(1)$
Машинное слово	$O(1)$	$O(1)$

Пояснения:

Для множества, представленного как массив символов, операции объединения и пересечения требуют сравнения всех пар элементов, что даёт временную сложность порядка $O(n^2)$. Фактические замеры времени выполнения подтверждают эту сложность: операции над массивом символов требуют перебора всех элементов, что пропорционально n^2.

Для множества, представленного в виде связного списка, для выполнения операции пересечения необходимо сравнить каждый элемент первого списка с каждым элементом второго списка. Аналогично операциям объединения: необходимо пройти по всем элементам, чтобы избежать дубликатов. Это также приводит к сложности $O(n^2)$.

Для множества, представленного как массив битов, операции пересечения и объединения выполняются с использованием побитовых операций, что значительно ускоряет вычисления. Ожидаемая сложность операций составляет $O(|U|)$, где $|\ U\ |$ — мощность универсума. Так как размер универсума ограничен (в данном случае 10 цифр), сложность фактически остаётся постоянной — $O(1)$.

Для множества, представленного в виде машинного слова, операции над множеством выполняются за константное время $O(1)$, так как все операции объединения и пересечения сводятся к простым битовым операциям над одним машинным словом.

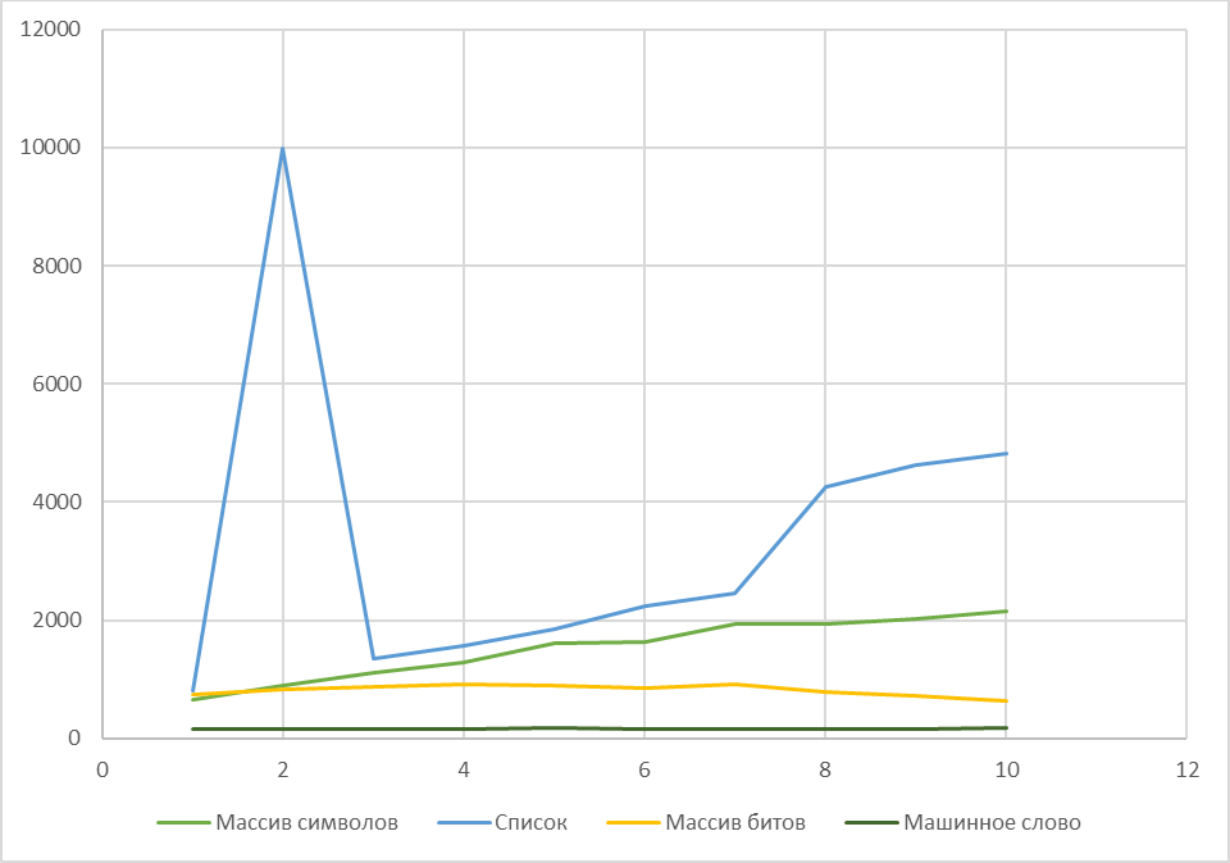
Результат измерения времени обработки для каждого из способов

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	*645*	*802*	*749*	*151*
2	*889*	*9986*	*823*	*148*
3	*1115*	*1352*	*873*	*151*
4	*1287*	*1568*	*905*	*157*

5	*1605*	*1840*	*897*	*171*
6	*1630*	*2233*	*858*	*165*
7	*1931*	*2456*	*917*	*148*
8	*1936*	*4264*	*795*	*160*
9	*2028*	*4616*	*721*	*161*
10	*2157*	*4815*	*638*	*182*

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 результате работы проведён анализ эффективности четырёх способов представления множеств в памяти ЭВМ: массива символов, связного списка, массива битов и машинного слова. В ходе исследования выявлено, что наиболее эффективным способом представления множества для быстрого выполнения операций объединения и пересечения является машинное слово.

Использование машинного слова показало минимальное время выполнения операций. Это обусловлено тем, что все операции над множествами сводятся к простым побитовым операциям, выполняемым на уровне процессора. Машинное слово рекомендуется использовать, когда мощность универсума небольшая и не превышает разрядность машинного слова (например, при представлении множества десятичных цифр).

Использование массива битов также демонстрирует высокую производительность. Данный способ позволяет выполнять операции объединения и пересечения с использованием побитовых операций над массивами, что обеспечивает сложность $O(|U|)$, где $|U|$ — мощность универсума. В данной работе это константное значение (размер универсума — 10), поэтому фактическая сложность операций приближается к $O(1)$. Однако эффективность этого метода снижается, если мощность универсума значительно возрастает или становится неопределённой.

Представление множества в виде массива символов оказалось менее производительным по сравнению с машинным словом и массивом битов. Операции объединения и пересечения требуют сравнения всех пар элементов, что приводит к сложности $O(n^2)$. Однако этот способ может быть полезен в ситуациях, когда можно точно определить размер массива и мощность универсума слишком велика для использования массива битов или машинного слова.

Связный список оказался самым медленным способом представления множества. Данный метод требует дополнительного времени для

динамического выделения памяти под узлы списка и обхода всех элементов при выполнении операций объединения и пересечения. В результате сложность операций составляет $O(n^2)$, что делает связный список непригодным для обработки больших множеств. Тем не менее, данный способ целесообразно использовать в случаях, когда изначально неизвестен размер множества, а выделение памяти под все элементы невозможно.

Список используемых источников

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

2. Алгоритмы и структуры данных. Лекции на сайте https://vec.etu.ru/moodle/course/view.php?id=13286.

Приложение

Код Array.cpp

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

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

using namespace std;

const int U = 10;

void scan_set(char t[U + 1]) {
    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[U + 1]{}, b[U + 1]{}, c[U + 1]{}, d[U + 1]{}, e[U + 1]{};
    int cnt_e = 0;
    bool found;

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

    auto start = chrono::high_resolution_clock::now();

    // e = a & b
    for (int i = 0; i <= U && a[i] != '\000'; ++i) {
        for (int j = 0; j <= U && b[i] != '\000'; ++j) {
            if (a[i] == b[j]) {
                e[cnt_e++] = a[i];
                break;
            }
        }
    }

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

```
    // e |= d
    for (int i = 0; i <= U && d[i] != '\000'; ++i) {
        found = false;
        for (int j = 0; j < cnt e; ++j) {
            if (e[j] == d[i]) {
                found = true;
                break;
            }
        }
        if (!found) {
            e[cnt e++] = d[i];
        }
    }
    auto stop = chrono::high resolution clock::now();

    // print
    if (loc) cout << "E: ";
    for (int i = 0; i < cnt e; ++i) {
        cout << e[i] << " ";
    }
    cout << "in " << chrono::duration cast<chrono::nanoseconds>(stop - start).count() << "
nanoseconds";

    return 0;
}
```

Код List.cpp

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

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

using namespace std;

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) {
    for (Node *cur = *head; cur != nullptr; cur = cur->next) {
        if (cur->data == data) {
            return; // Node already exists
        }
    }
    Node *newNode = createNode(data);
    newNode->next = *head;
    *head = 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) {
    for (Node *curA = headA; curA != nullptr; curA = curA->next) {
        bool found = false;
        for (Node *curB = headB; curB != nullptr; curB = curB->next) {
            if (curA->data == curB->data) {
```

```
                found = true;
                break;
            }
        }
        if (found) {
            insertNode(result, curA->data);
        }
    }
}

void bitwiseOr(Node *headA, Node **result) {
    for (Node *cur = headA; cur != nullptr; cur = cur->next) {
        insertNode(result, cur->data);
    }
}

void deleteList(Node **head) {
    Node *cur = *head;
    while (cur != nullptr) {
        Node *next = cur->next;
        delete cur;
        cur = next;
    }
    *head = nullptr;
}

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

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

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

    // print
    if (loc) cout << "E: ";
    Node *cur = headE;
    while (cur != nullptr) {
        cout << cur->data << " ";
        cur = cur->next;
    }
    cout << "in " << chrono::duration_cast<chrono::nanoseconds>(stop - start).count() << "
nanoseconds";

    // deleting lists
    deleteList(&headA);
    deleteList(&headB);
    deleteList(&headC);
    deleteList(&headD);
    deleteList(&headE);

    return 0;
}
```

Код Bits.cpp

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

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

```
    bool loc = false;
#endif

using namespace std;

const int U = 10;

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

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

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

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

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

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

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

    return 0;
}
```

Код MWord.cpp

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

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

using namespace std;

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) cout << "A: ";
    a = scan_set();
    if (loc) cout << "B: ";
    b = scan_set();
    if (loc) cout << "C: ";
    c = scan_set();
    if (loc) cout << "D: ";
    d = scan_set();

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

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

    return 0;
}
```

Код Generator.cpp

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

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

using namespace std;

void generator(char a[], int &len) {
    mt19937 rnd(random_device{}());
    len = 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 (uniform_int_distribution<int>(0, 1)(rnd)) {
            a[i] = digits[i];
        } else {
            a[i] = '\0';
        }
    }

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

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

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

    for (int i = 9; i > 0; --i) {
        int j = uniform_int_distribution<int>(0, i)(rnd);
        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) {
        cout << "Enter length of generated sets (from 0 to 10):\n";
        cin >> len;
    } else {
        if (argc > 1) {
            len = 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') {
                cout << a[j] << " ";
            }
        }
        cout << "\n";
    }
    return 0;
}
```

Код AutoTest.cpp

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

using namespace std;

void out print(const char name[]) {
    string line;
    ifstream file(name);
    getline(file, line);
    file.close();
    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(format("Generator {} > input.txt", i).c str());
        else system("Generator > input.txt");

        cout << "Test " << i << ":\n";
        ifstream file("input.txt");
        string line;
        char sym = 'A';
        while (getline(file, line)) {
```

```
            cout << sym++ << ": " << line << "\n";
        }
        file.close();

        system("Array < input.txt > out array.txt");
        cout << "Array result: ";
        out print("out array.txt");

        system("List < input.txt > out list.txt");
        cout << "\nList result:  ";
        out print("out list.txt");

        system("Bits < input.txt > out bits.txt");
        cout << "\nBits result:  ";
        out print("out bits.txt");

        system("MWord < input.txt > out mword.txt");
        cout << "\nMWord result: ";
        out print("out mword.txt");
        cout << "\n\n";
    }

    return 0;
}
```