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»

Выполнил: Барченков П. А.

Бужинский Д. С

Факультет: КТИ

Группа: №3312

Принял: старший преподаватель Колинько П. Г.

Подпись преподавателя: _____

Санкт-Петербург

2024

Содержание

Цель работы

Сравнительное исследование четырёх способов хранения множеств в памяти ЭВМ.

Задание

Множество, содержащее буквы, имеющиеся во множестве A, но не являющиеся общими для B и C, и все буквы из D.

Формула для вычисления пятого множества

Формализация задания: $E = A \setminus (B \cap C) \cup D$

Контрольные тесты

Ниже представлены примеры работы программы (рис. 1 – 4) . Используя генератор подмножеств заданной мощности, формируем подмножества с мощностью от 2 до 52 и считаем количество микросекунд, произошедших за период обработки подмножеств в различных формах их представления. Результат выводится в консоль.

```
C:\Users\79627\Documents\Alg\main.exe
Count:2
A = H I
B = x R
C = Y T
D = c h
LIST
E = H I c h
Time 10200 nanoseconds
ARRAY
E = H I c h
Time 1300 nanoseconds
MACHINE WORDS
A = 00000000000000000110000000000000000000000000000000000000 (H I )
B = 00000000100000000000000000000010000000000000000000000000 (x R )
C = 01000010000000000000000000000000000000000000000000000000 (T Y )
D = 00000000000000000000000000000000000000000000000010000100 (c h )
E = 00000000000000000110000000000000000000000000000010000100 (c h H I )
Time 100 nanoseconds
UNIVERSE REPRESENTATION
A = (H I )
B = (x R )
C = (T Y )
D = (c h )
E = (c h H I )
Time 600 nanoseconds

Process finished with exit code 0
```

Рис. 1

```
C:\Users\79627\Documents\Alg\main.exe
Count:3
A = A B C
B = E B C
C = A d C
D = A M x
LIST
E = A B M x
Time 10100 nanoseconds
ARRAY
E = A B M x
Time 1300 nanoseconds
MACHINE WORDS
A = 00000000000000000000000111000000000000000000000000000 (A B C )
B = 00000000000000000000010110000000000000000000000000000 (B C E )
C = 00000000000000000000001010000000000000000000000001000 (d A C )
D = 00000000000001000000000001001000000000000000000000000 (x A M )
E = 00000000000001000000000011001000000000000000000000000 (x A B M )
Time 100 nanoseconds
UNIVERSE REPRESENTATION
A = (A B C )
B = (B C E )
C = (d A C )
D = (x A M )
E = (x A B M )
Time 800 nanoseconds

Process finished with exit code 0
```

Рис. 2

```
C:\Users\79627\Documents\Alg\main.exe
Count:4
A = A B C d
B = E A C g
C = A d C v
D = A M x R
LIST
E = B d A M x R
Time 2300 nanoseconds
ARRAY
E = B d A M x R
Time 1400 nanoseconds
MACHINE WORDS
A = 00000000000000000000000111000000000000000000000001000 (d A B C )
B = 00000000000000000000010101000000000000000000001000000 (g A C E )
C = 00000000000000000000001010000100000000000000000001000 (d v A C )
D = 00000000100001000000000001001000000000000000000000000 (x A M R )
E = 00000000100001000000000011001000000000000000000001000 (d x A B M R )
Time 100 nanoseconds
UNIVERSE REPRESENTATION
A = (d A B C )
B = (g A C E )
C = (d v A C )
D = (x A M R )
E = (d x A B M R )
Time 800 nanoseconds

Process finished with exit code 0
```

Рис. 3

```
C:\Users\79627\Documents\Alg\main.exe
Count:10
A = R r S Q J w a h X P
B = M o V m F v E l h g
C = Y r W G D n o n Z t
D = X a b u c y v j V N
LIST
E = R r S Q J w a h X P b u c y v j V N
Time 6100 nanoseconds
ARRAY
E = R r S Q J w a h X P b u c y v j V N
Time 2900 nanoseconds
MACHINE WORDS
A = 00100001111000001000000000001000010000000010000001 (a h r w J P Q R S X )
B = 00001000000001000000110000000010000001011000011000000 (g h l m o v E F M V )
C = 11010000000000000001001000000000101001100000000000000 (n o r t D G W Y Z )
D = 00101000000010000000000000010011000000000001000000111 (a b c j u v y N V X )
E = 00101001111010001000000000010111001000000001010000111 (a b c h j r u v w y J N P Q R S V X )
Time 100 nanoseconds
UNIVERSE REPRESENTATION
A = (a h r w J P Q R S X )
B = (g h l m o v E F M V )
C = (n o r t D G W Y Z )
D = (a b c j u v y N V X )
E = (a b c h j r u v w y J N P Q R S V X )
Time 1000 nanoseconds

Process finished with exit code 0
```

Рис. 4

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		$O(n^3)$
Универсум	$O(\|U\|)$	$O(\|U\|)$
Машинное слово	$O(1)$	$O(1)$

Пояснения:

Для обработки массива символов и списка требуется проверка всех комбинаций элементов множества, которых для множеств мощностью n будет $O(n^2)$. Поэтому временная сложность будет квадратичной $O(n^2)$, что совпадает с фактической сложностью.

Для множеств, представленных отображением на универсум ожидаемое количество шагов мощности универсума, и оно совпадает с фактической.

Операции с множествами в виде машинного слова выполняются за один шаг независимо от мощности множеств и имеют временную сложность O(1).

Результат измерения времени обработки для каждого из способов

Таблица 2. Результаты измерения времени в наносекундах

Мощность множеств	Количество наносекунд, проходящих за время обработки множеств при различных способах представления			
	Список	Массив символов	Машинное слово	Массив битов
2	10200	800	100	700
6	2600	1200	100	700
10	5500	2100	100	900
14	4000	2500	100	1000
18	7500	3100	100	1200
22	11600	3800	100	1300
26	9900	4900	100	1400
30	12400	5900	100	1500
34	22600	6000	100	1400
38	24100	7100	100	1400

42	23300	7400	100	1300
46	23200	8600	100	1500
50	25100	8600	100	1500
52	30100	7900	100	1200

При представлении множеств в виде массива символов и списка заметно, что время обработки множеств с увеличением мощности также увеличивается. Для универсума и машинного слова время обработки практически неизменно, т.е. не зависит от размера входа для машинного слова, и немного увеличивается для массива битов.

Выводы

В ходе данной работы выяснили, что наиболее быстрым способом представления множеств для их обработки является машинное слово. Этот способ рекомендуется использовать, когда есть простая функция для отображения элемента множества в соответствующий ему порядковый номер бита и размер универсума не превышает разрядности слова.

Самой медленной оказалась обработка списков. Данный способ представления нужно использовать, когда мощность создаваемого множества неизвестна, и поэтому выделить память сразу под всё множество невозможно.

Представлять множество в виде набора элементов в массиве следует в том случае, если можно с достаточной точностью знать размер массива, а мощность универсума слишком велика для использования вектора битов или машинного слова.

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. Множества в памяти ЭВМ // Алгоритмы и структуры данных. Лекция от 10.09.2020.

3. Студент Сабиров Р. Частное сообщение.

Приложение. Текст программы

```
#include <iostream>
#include <bitset>
#include <ctime>
#include <chrono>

const int max_size = 60;
const int UNIVERSE_SIZE = 52;
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 insertAtEnd(Node **head, char data) {
    Node *newNode = createNode(data);
    if (*head == nullptr) {
        *head = newNode;
        return;
    }
    Node *temp = *head;
    while (temp->next != nullptr) {
        temp = temp->next;
    }
    temp->next = newNode;
}

Node *arrayToList(const char arr[], int size) {
    Node *head = nullptr;
    for (int i = 0; i < size; i++) {
        if (arr[i] != 0) {
            insertAtEnd(&head, arr[i]);
        }
    }
    return head;
}

void printList(Node *head) {
    Node *temp = head;
    while (temp != nullptr) {
        cout << temp->data << " ";
        temp = temp->next;
    }
    cout << endl;
```

```
}

Node *setUnion(Node *A, Node *B) {
    Node *result = nullptr;
    Node *tempA = A;

    while (tempA != nullptr) {
        insertAtEnd(&result, tempA->data);
        tempA = tempA->next;
    }

    Node *tempB = B;
    while (tempB != nullptr) {
        bool found = false;
        Node *tempResult = result;
        while (tempResult != nullptr) {
            if (tempB->data == tempResult->data) {
                found = true;
                break;
            }
            tempResult = tempResult->next;
        }
        if (!found) {
            insertAtEnd(&result, tempB->data);
        }
        tempB = tempB->next;
    }

    return result;
}

Node *setIntersection(Node *A, Node *B) {
    Node *result = nullptr;
    Node *tempA = A;

    while (tempA != nullptr) {
        Node *tempB = B;
        while (tempB != nullptr) {
            if (tempA->data == tempB->data) {
                insertAtEnd(&result, tempA->data);
                break;
            }
            tempB = tempB->next;
        }
        tempA = tempA->next;
    }

    return result;
}

Node *setDifference(Node *A, Node *B) {
    Node *result = nullptr;
    Node *tempA = A;

    while (tempA != nullptr) {
        bool found = false;
        Node *tempB = B;
        while (tempB != nullptr) {
            if (tempA->data == tempB->data) {
                found = true;
                break;
            }
```

```
            tempB = tempB->next;
        }
        if (!found) {
            insertAtEnd(&result, tempA->data);
        }
        tempA = tempA->next;
    }

    return result;
}

//                                                  ДЛЯ МАССИВОВ

void Union(const char A[], const char B[], char res[]) {
    int k = 0;
    for (int i = 0; i < A[i] != 0; i++) {
        res[k++] = A[i];
    }
    for (int i = 0; i < B[i] != 0; i++) {
        bool fl = false;
        for (int j = 0; j < k; j++) {
            if (B[i] == res[j]) {
                fl = true;
                break;
            }
        }
        if (fl == false) {
            res[k++] = B[i];
        }
    }
}

void Intersection(const char A[], const char B[], char res[]) {
    int k = 0;
    for (int i = 0; A[i] != 0; i++) {

        for (int j = 0; j < max_size; j++) {
            if (A[i] == B[j]) {
                res[k++] = A[i];
                break;
            }
        }
    }
}

void Difference(const char A[], const char B[], char res[]) {
    int k = 0;
    for (int i = 0; i < A[i] != 0; i++) {
        bool fl = false;
        for (int j = 0; j < max_size; j++) {
            if (A[i] == B[j]) {
                fl = true;
                break;
            }
        }
        if (fl == false) {
            res[k++] = A[i];
        }
    }
}

void print(char set[]) {
```

```
    for (int i = 0; i < set[i] != 0; i++) {
        cout << set[i] << " ";
    }
    cout << endl;
}
//                                  МАШИННОЕ СЛОВО

unsigned long long arrayToMachineWord(const char arr[], int size) {
    unsigned long long word = 0;
    for (int i = 0; i < size; i++) {
        if (arr[i] != 0) {
            int charIndex;
            if (arr[i] >= 'a' && arr[i] <= 'z') {
                charIndex = arr[i] - 'a';
            } else if (arr[i] >= 'A' && arr[i] <= 'Z') {
                charIndex = arr[i] - 'A' + 26;
            }
            word |= (1ULL << charIndex);
        }
    }
    return word;
}

void printMachineWord(unsigned long long word) {
    bitset<UNIVERSE_SIZE> bits(word);
    cout << bits << " (";
    for (int i = 0; i < UNIVERSE_SIZE; i++) {
        if (bits[i]) {
            if (i < 26) {
                cout << (char) ('a' + i) << ' ';
            } else {
                cout << (char) ('A' + i - 26) << ' ';
            }
        }
    }
    cout << ")" << endl;
}

//                      УНИВЕРСУМ


void mapToUniverse(const char *str, bool *bitVector) {
    for (int i = 0; i < UNIVERSE_SIZE; ++i)
        bitVector[i] = false;

    for (int i = 0; str[i]; ++i) {
        int index;
        if (islower(str[i])) {
            index = str[i] - 'a';
        } else {
            index = str[i] - 'A' + 26;
        }
        if (index >= 0 && index < UNIVERSE_SIZE) {
            bitVector[index] = true;
        }
    }
}

void unionSets(const bool *A, const bool *B, bool *result) {
    for (int i = 0; i < UNIVERSE_SIZE; ++i)
        result[i] = A[i] || B[i];
}
```

```
void differenceSets(const bool *A, const bool *B, bool *result) {
    for (int i = 0; i < UNIVERSE_SIZE; ++i)
        result[i] = A[i] && !B[i];
}

void intersectionSets(const bool *A, const bool *B, bool *result) {
    for (int i = 0; i < UNIVERSE_SIZE; ++i)
        result[i] = A[i] && B[i];
}

void printUniverse(const bool *bitVector) {
    cout << "(";
    for (int i = 0; i < UNIVERSE_SIZE; i++) {
        if (bitVector[i]) {
            if (i < 26) {
                cout << (char) (L'a' + i) << ' ';
            } else {
                cout << (char) (L'A' + (i - 26)) << ' ';
            }
        }
    }
    cout << ")" << endl;
}

int n;

void generateArray(char arr[], int size) {
    for (int i = 0; i < n/*(rand()%49)+3*/; i++) {
        if (rand() % 2) {
            arr[i] = 'a' + rand() % 26;
        } else {
            arr[i] = 'A' + rand() % 26;
        }

    }
}

int main() {
    cout << "Count:";

    cin >> n;
    srand(time(0));
    char A_arr[max_size] = {0};
    char B_arr[max_size] = {0};
    char C_arr[max_size] = {0};
    char D_arr[max_size] = {0};

    generateArray(A_arr, max_size);
    generateArray(B_arr, max_size);
    generateArray(C_arr, max_size);
    generateArray(D_arr, max_size);

    Node *A = arrayToList(A_arr, max_size);
    Node *B = arrayToList(B_arr, max_size);
    Node *C = arrayToList(C_arr, max_size);
    Node *D = arrayToList(D_arr, max_size);

    cout << "A = ";
    printList(A);
    cout << "B = ";
    printList(B);
```

```
    cout << "C = ";
    printList(C);
    cout << "D = ";
    printList(D);

    auto list_t1 = chrono::high_resolution_clock::now();
    Node *rez1 = setIntersection(B, C);
    Node *rez2 = setDifference(A, rez1);
    Node *E = setUnion(rez2, D);
    auto list_t2 = chrono::high_resolution_clock::now();
    auto list_time_res = chrono::duration_cast<chrono::duration<double,
nano>>(list_t2 - list_t1).count();
    cout << "LIST \n";
    cout << "E = ";
    printList(E);
    cout << "Time " << list_time_res << " nanoseconds\n";
    char res_arr1[max_size] = {0};
    char res_arr2[max_size] = {0};
    char E_arr[max_size] = {0};
    auto arr_t1 = chrono::high_resolution_clock::now();
    Intersection(B_arr, C_arr, res_arr1);
    Difference(A_arr, res_arr1, res_arr2);
    Union(res_arr2, D_arr, E_arr);
    auto arr_t2 = chrono::high_resolution_clock::now();
    auto arr_time_res = chrono::duration_cast<chrono::duration<double,
nano>>(arr_t2 - arr_t1).count();

    cout << "ARRAY \n";
    cout << "E = ";
    print(E_arr);
    cout << "Time " << arr_time_res << " nanoseconds\n";
    unsigned long long A_word = arrayToMachineWord(A_arr, max_size);
    unsigned long long B_word = arrayToMachineWord(B_arr, max_size);
    unsigned long long C_word = arrayToMachineWord(C_arr, max_size);
    unsigned long long D_word = arrayToMachineWord(D_arr, max_size);

    cout << "MACHINE WORDS \n";
    cout << "A = ";
    printMachineWord(A_word);
    cout << "B = ";
    printMachineWord(B_word);
    cout << "C = ";
    printMachineWord(C_word);
    cout << "D = ";
    printMachineWord(D_word);
    auto mw_t1 = chrono::high_resolution_clock::now();
    unsigned long long res_word1 = (B_word & C_word);
    unsigned long long res_word2 = (A_word & ~(res_word1));
    unsigned long long E_word = (res_word2 | D_word);
    auto mw_t2 = chrono::high_resolution_clock::now();
    auto mw_time_res = chrono::duration_cast<chrono::duration<double,
nano>>(mw_t2 - mw_t1).count();
    cout << "E = ";
    printMachineWord(E_word);
    cout << "Time " << mw_time_res << " nanoseconds\n";

    //              УНИВЕРСУМ
    bool bitA[UNIVERSE_SIZE], bitB[UNIVERSE_SIZE], bitC[UNIVERSE_SIZE],
bitD[UNIVERSE_SIZE];
    bool bit_res1[UNIVERSE_SIZE], bit_res2[UNIVERSE_SIZE],
E_bit[UNIVERSE_SIZE];
```

```
    // Отображаем строки на векторы битов
    mapToUniverse(A_arr, bitA);
    mapToUniverse(B_arr, bitB);
    mapToUniverse(C_arr, bitC);
    mapToUniverse(D_arr, bitD);

    cout << "UNIVERSE REPRESENTATION \n";
    cout << "A = ";
    printUniverse(bitA);
    cout << "B = ";
    printUniverse(bitB);
    cout << "C = ";
    printUniverse(bitC);
    cout << "D = ";
    printUniverse(bitD);
    auto uni_t1 = chrono::high_resolution_clock::now();
    intersectionSets(bitB, bitC, bit_res1);
    differenceSets(bitA, bit_res1, bit_res2);
    unionSets(bit_res2, bitD, E_bit);
    auto uni_t2 = chrono::high_resolution_clock::now();
    auto uni_time_res = chrono::duration_cast<chrono::duration<double,
nano>>(uni_t2 - uni_t1).count();
    cout << "E = ";
    printUniverse(E_bit);
    cout << "Time " << uni_time_res << " nanoseconds\n";
    return 0;
}
```