**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра Вычислительной техники**

отчет

**по лабораторной работе № 2**

**по дисциплине «Алгоритмы и структуры данных»**

Тема: **«Множество как объект»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 3311 | Баймухамедов Р. Р. |  |
| Преподаватель | Манирагена В. |  |

Санкт-Петербург

2024

**Введение**

**Цель работы:**

Исследование эффекта от использования класса при хранении множества в памяти ЭВМ

Исследование четырёх способов хранения множеств в памяти ЭВМ

**Задание на обработку множеств:** (Вариант 21 - Десятичные цифры):

Множество, содержащее цифры, имеющиеся в *A* или в *B* или являющиеся общими для *C* и *D*

**Формализация задания:**

E = A ∪ B ∪ (С ∩ D)

**Контрольные тесты:**

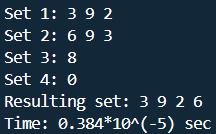


Рисунок 1 – Массивы

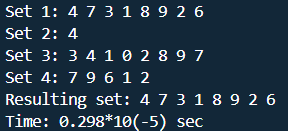


Рисунок 2 – Списки

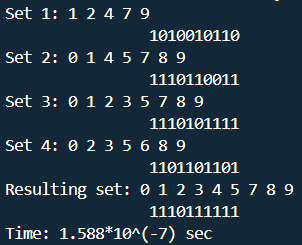


Рисунок 3 – Массивы битов

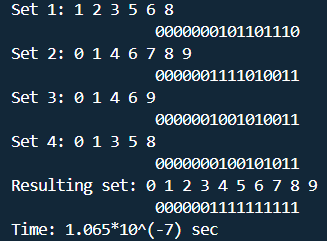


Рисунок 4 – Машинное слово

Выше представлены контрольные примеры, использующие разные способы хранения множеств в памяти ЭВМ, реализованные как объекты класса Set. Данные о множествах были сгенерированы случайным образом. На рис. 3 и рис. 4 также представлен ответ в виде массивов битов и машинного слова (двоичного кода) соответственно

**Результаты эксперимента с четырьмя структурами данных на основе классов**

Временная сложность алгоритмов обработки сохраняется в соответствии с лабораторной работой №1. Аналогично достоинства, недостатки и области применения.

Приведем время выполнения алгоритма с множествами, представленными как объектами

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Номер попытки | Время выполнения алгоритма для представленного способа (сек) | | | |
| Массивы | Списки | Массивы битов | Машинное слово |
| 1 | 0,267\*10-5 | 0,298\*10-5 | 0,144\*10-6 | 0,105\*10-6 |
| 2 | 0,314\*10-5 | 0,287\*10-5 | 0,143\*10-6 | 0,104\*10-6 |
| 3 | 0,460\*10-5 | 0,285\*10-5 | 0,144\*10-6 | 0,103\*10-6 |
| 4 | 0,379\*10-5 | 0,284\*10-5 | 0,172\*10-6 | 0,112\*10-6 |
| 5 | 0,317\*10-5 | 0,313\*10-5 | 0,147\*10-6 | 0,113\*10-6 |
| 6 | 0,467\*10-5 | 0,296\*10-5 | 0,170\*10-6 | 0,100\*10-6 |
| 7 | 0,326\*10-5 | 0,306\*10-5 | 0,142\*10-6 | 0,114\*10-6 |
| 8 | 0,365\*10-5 | 0,304\*10-5 | 0,161\*10-6 | 0,103\*10-6 |
| 9 | 0,365\*10-5 | 0,292\*10-5 | 0,142\*10-6 | 0,121\*10-6 |
| 10 | 0,459\*10-5 | 0,284\*10-5 | 0,172\*10-6 | 0,106\*10-6 |
| Среднее значение | 0,372\*10-5 | 0,295\*10-5 | 0,153\*10-6 | 0,108\*10-6 |

**Результат эксперимента с отслеживанием вызовов функций-членов**

**Заключение:**

В ходе выполнения задания успешно были применены операции над множествами для различных типов представления данных: списков, массивов, массивов битов и машинной формы. Мы провели объединение и пересечение множеств, сформировав итоговое множество, которое содержит уникальные элементы из множеств A и B, а также общие элементы из множеств C и D.

Для каждого представления были реализованы соответствующие алгоритмы, что позволило укрепить навыки работы с различными структурами данных и их эффективной обработки. Применение списков продемонстрировало гибкость и наглядность работы с последовательностями, массивы позволили эффективно управлять структурой данных с фиксированным размером, битовые операции показали преимущества работы с данными на побитовом уровне, что особенно полезно в задачах с ограниченными ресурсами, а машинная форма обеспечила минимизацию пространства и максимальную скорость выполнения операций за счёт низкоуровневого представления данных.

Решение задачи позволило продемонстрировать эффективность использования различных подходов к работе с множествами для решения практических задач по объединению, пересечению и фильтрации данных. Сравнение времени выполнения операций для различных представлений множеств даёт наглядное понимание того, как выбор структуры данных влияет на производительность и эффективность программы в целом.

**Текст программ**

Array.h

#include <vector>

#include <string>

class Set {

private:

    std::vector<int> set;

    std::string name;

public:

    int size() *const*;

    void set\_the\_name(std::string name);

    void input\_in\_set();

    void print\_set() *const*;

    bool is\_in\_set(int element) *const*;

    void association(*const* Set*&* additional);

    void intersection(*const* Set*&* additional);

    void fill\_random();

    void timer();

};

Array.cpp

#include "Array.h"

#include <iostream>

#include <vector>

#include <sstream>

#include <string>

#include <limits>

int Set::size() *const*{

    return set.size();

}

void Set::set\_the\_name(std::string name){

    this->name = name;

}

void Set::input\_in\_set(){

    std::string input;

    std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), '\n'); *// Очистка буфера ввода*

    std::cout<<"Enter the numbers of " << name << " separated by space: ";

    std::getline(std::cin, input);

    std::stringstream ss(input);

    int number;

    while(ss >> number){

        set.push\_back(int(number));

    }

}

void Set::print\_set() *const*{

    std::cout << name << ": ";

    for(int i=0; i<set.size();i++){

        std::cout << set[i] << " ";

    }

    std::cout << std::endl;

}

bool Set::is\_in\_set(int element) *const* {

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

        if (element == set[i]) return true;

    }

    return false;

}

void Set::association(*const* Set*&* additional) {

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

        if (!is\_in\_set(additional.set[i])) {

            set.push\_back(additional.set[i]);

        }

    }

}

void Set::intersection(*const* Set*&* additional) {

    auto number = set.begin();

    while (number != set.end()) {

        if (!additional.is\_in\_set(\*number)) {

            number = set.erase(number); *// Delete the value of set*

        } else {

            number++;

        }

    }

}

void Set::fill\_random(){

    for (int i = 0; i < (rand() % 10); i++) {

        int num = rand() % 10;

        if(!is\_in\_set(num)) set.push\_back(num);

    }

}

void Set::timer(){

    int t1 = clock();

    for(int i=0; i<100000;i++){

        std::vector<Set> sets(5);

        for(int i=0; i<4;i++){

            sets[i].set\_the\_name("Set "+std::to\_string(i+1));

            sets[i].fill\_random();

        }

        sets[4] = sets[0];

        sets[4].association(sets[1]);

        sets[2].intersection(sets[3]);

        sets[4].association(sets[2]);

    }

    int t2 = clock();

    double t = double(t2 - t1) / CLOCKS\_PER\_SEC;

    std::cout << "Time: " << t << "\*10^(-5) sec";

}

List.h

#include <vector>

#include <string>

struct Node{ *// Structure of Node*

        int data;

        Node \*next;

};

class Set {

private:

    Node \*set = nullptr;

    std::string name;

public:

    Set();

    void delete\_list();

    ~Set();

    void push\_back(int value);

    void set\_the\_name(std::string name);

    void input\_in\_set();

    void print\_set();

    bool is\_in\_set(int element);

    void association(*const* Set *&*additional);

    void intersection(Set*&* additional);

    void fill\_random();

    void timer();

};

List.cpp

#include "List.h"

#include <iostream>

#include <vector>

#include <sstream>

#include <string>

#include <limits>

#include <time.h>

Set::Set() : set(nullptr) {}

void Set::delete\_list() {

    while (set) {

        Node\* temp = set;

        set = set->next;

        delete temp;

    }

}

Set::~Set() {

    delete\_list();

}

void Set::push\_back(int value) { *// Function of adding the last elem in list*

    Node\* newNode = new Node{ value, nullptr }; *// Create new node*

    if (!set) { *// If list is empty*

        set = newNode;

    }

    else { *// If list is not empty*

        Node\* current = set;

        while (current->next) { *// Find the last elem of list*

            current = current->next;

        }

        current->next = newNode;

    }

}

void Set::set\_the\_name(std::string name){

    this->name = name;

}

void Set::print\_set() {

    Node\* current = set; *// Создаем временный указатель, чтобы не изменить set*

    std::cout << name << ": ";

    while (current) {

        std::cout << current->data << " ";

        current = current->next;

    }

    std::cout << std::endl;

}

bool Set::is\_in\_set(int element){

    Node\* current = set; *// Создаем временный указатель*

    while (current) {

        if (element == current->data) {

            return true;

        }

        current = current->next;

    }

    return false;

}

void Set::association(*const* Set*&* additional) {

    Node\* current = additional.set;

    while (current) {

        if (!is\_in\_set(current->data)) {

            push\_back(current->data);

        }

        current = current->next;

    }

}

void Set::intersection(Set*&* additional) {

    Node\* current = set;

    Node\* prev = nullptr;

    while (current) {

        if (!additional.is\_in\_set(current->data)) {

            if (prev) {

                prev->next = current->next;

            }

            else {

                set = current->next;

            }

            delete current;

            current = prev ? prev->next : set;

        }

        else {

            prev = current;

            current = current->next;

        }

    }

}

void Set::fill\_random() {

    int count = rand() % 10;

    while (count > 0) {

        int num = rand() % 10;

        if (!is\_in\_set(num)) {

            this->push\_back(num);

            count--;

        }

    }

}

void Set::timer() {

    srand(time(0)); *// Инициализация генератора случайных чисел*

    int t1 = clock();

    for (int i = 0; i < 100000; i++) {

        Set \*A\_list = new Set;

        Set \*B\_list = new Set;

        Set \*C\_list = new Set;

        Set \*D\_list = new Set;

        Set \*CiD\_list = new Set;

        Set \*E\_list = new Set;

        A\_list->fill\_random();

        B\_list->fill\_random();

        C\_list->fill\_random();

        D\_list->fill\_random();

        E\_list->association(\*A\_list);

        E\_list->association(\*B\_list);

        CiD\_list->intersection(\*C\_list);

        E\_list->association(\*CiD\_list);

    }

    int t2 = clock();

    double t = double(t2 - t1) / CLOCKS\_PER\_SEC;

    std::cout << "Time: " << t << "\*10(-5) sec" << std::endl;

}

Barray.h

#include <vector>

#include <string>

#include <bitset>

#define SIZE 10

class Set {

private:

    std::bitset<SIZE> set;

    std::string name;

public:

    void set\_the\_name(std::string name);

    void print\_set() *const*;

    void association(*const* Set*&* additional);

    void intersection(*const* Set*&* additional);

    void fill\_random();

    void timer();

};

Barray.cpp

#include "Barray.h"

#include <iostream>

#include <vector>

#include <time.h>

#include <bitset>

#define SIZE 10

void Set::set\_the\_name(*const* std::string name){

    this->name = name;

}

void Set::association(*const* Set*&* additional) {

    set |= additional.set; *// Побитовое ИЛИ для объединения*

}

void Set::intersection(*const* Set*&* additional) {

    set &= additional.set; *// Побитовое И для пересечения*

}

void Set::print\_set() *const*{

    std::cout << name << ": ";

    for(int i=0; i<10; i++){

        if(set[i]==1){

            std::cout << i << " ";

        }

    }

    std::cout << "\n\t\t";

    for(int i=9; i>=0; i--){

        if(set[i]==1) std::cout << 1;

        else std::cout << 0;

    }

    std::cout << std::endl;

}

void Set::fill\_random(){

    set = std::bitset<SIZE> (rand() % 1024);

}

void Set::timer(){

    srand(time(0)); *// To random values*

    int t1 = clock();

    for(int i=0; i<10000000; i++){

        Set bA, bB, bC, bD, bE;

        bA.fill\_random();

        bB.fill\_random();

        bC.fill\_random();

        bD.fill\_random();

        bE = bA;

        bE.association(bB);

        bC.intersection(bD);

        bE.association(bC);

    }

    int t2 = clock();

    double t = double(t2-t1) / CLOCKS\_PER\_SEC;

    std::cout << "Time: " << t << "\*10^(-7) sec";

}

Mword.h

#include <vector>

#include <string>

class *Set* {

private:

    unsigned short set;

    std::*string* name;

public:

    void set\_the\_name(std::*string* name);

    void print\_set() *const*;

    void association(*const* *Set&* additional);

    void intersection(*const* *Set&* additional);

    void fill\_random();

    void timer();

};

Mword.cpp

#include "Mword.h"

#include <iostream>

#include <vector>

#include <time.h>

void Set::set\_the\_name(*const* std::string name){

    this->name = name;

}

void Set::association(*const* Set*&* additional) {

    set |= additional.set; *// Побитовое ИЛИ для объединения*

}

void Set::intersection(*const* Set*&* additional) {

    set &= additional.set; *// Побитовое И для пересечения*

}

void Set::print\_set() *const*{

    std::cout << name << ": ";

    for(int i=0; i<10; i++){

        if(set & (1 << i)){

            std::cout << i << " ";

        }

    }

    std::cout << "\n\t\t";

    for(int i=sizeof(set)\*8-1; i>=0; i--){

        if(set & (1 << i)){

            std::cout << 1;

        } else std::cout << 0;

    }

    std::cout <<std::endl;

}

void Set::fill\_random(){

    set = rand()%0x3FF;

}

void Set::timer(){

    srand(time(0)); *// To random values*

    int t1 = clock();

    for(int i=0; i<10000000; i++){

        Set wA, wB, wC, wD, wE;

        wA.fill\_random();

        wB.fill\_random();

        wC.fill\_random();

        wD.fill\_random();

        wE = wA;

        wE.association(wB);

        wC.intersection(wD);

        wE.association(wC);

    }

    int t2 = clock();

    double t = double(t2-t1) / CLOCKS\_PER\_SEC;

    std::cout << "Time: " << t << "\*10^(-7) sec";

}