Министерство науки и образования РФ

Федеральное государственное автономное образовательное

учреждение высшего профессионального образования

«Санкт-Петербургский государственный электротехнический

университет «ЛЭТИ» им. В. И. Ульянова (Ленина)»

(СПбГЭТУ «ЛЭТИ»)

Факультет компьютерных технологий и информатики

Кафедра вычислительной техники

**Отчёт по заданию № 1**

**на тему: “Множества”**

**по дисциплине**

**“Алгоритмы и структуры данных”**

**Вариант 26**

33

|  |  |
| --- | --- |
| Выполнил студент гр. 9308: | Яловега Н.В. |
| Проверил: | Колинько П.Г. |

Санкт-Петербург, 2020

Оглавление

[​ Введение 3](#__RefHeading___Toc4994_2043932214)

[​ 1. Задание 3](#__RefHeading___Toc4996_2043932214)

[​ 2. Формализация задания 3](#__RefHeading___Toc4998_2043932214)

[​ 3. Контрольные примеры 4](#__RefHeading___Toc5000_2043932214)

[​ 4. Временная сложность 5](#__RefHeading___Toc5002_2043932214)

[​ 5. Результаты измерения времени обработки 6](#__RefHeading___Toc5004_2043932214)

[​ 6. Результаты решения задачи 7](#__RefHeading___Toc5006_2043932214)

[​ Вывод 9](#__RefHeading___Toc5008_2043932214)

[​ Приложение 1 (Листинги программы) 10](#__RefHeading___Toc441_358905059)

## Введение

Целью задания является исследование четырех способов хранения множеств в памяти ЭВМ.

## **1. Задание**

Множество содержащие все буквы из A, не являющиеся общими для B, C и D.

Универсум — прописные латинские буквы.

## 2. Формализация задания

Условие задачи можно записать в виде формулы:

## **3. Контрольные примеры**

Таблица. 1. Контрольные примеры

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Исходные множества** | | | | **Результат** |
| **A** | **B** | **C** | **D** | **R** |
| 1 | ABCDFGHIJ | GHIJ | ABHIJ | ABFGHI | ABCDFGJ |
| 2 | ACDEKMOPRSUVWXZ | ABCDHIJLMNPQRWYZ | ACDFJKMNOSUVWYZ | BCDEGHILNPQRUVY | AEKMOPRSUVWXZ |
| 3 | ABCDFGHIJLMNQRSTUVZ | ABGJLMOQRSUY | ACEGHIJLMNOQTWZ | AFGJKLMPVWYZ | BCDFHINQRSTUVZ |
| 4 | ABC | DEF | GHI | XYZ | ABC |
| 5 | ABC | ABC | ABC | ABC | Пустое множество |

## **4. Временная сложность**

Таблица. 2. Временная сложность

|  |  |  |
| --- | --- | --- |
| **Способ представления** | **Ожидаемая** | **Фактическая** |
| Последовательность | O(1) | O(1) |
| Список | O(1) | O(1) |
| Машинное слово | O(1) | O(1) |
| Массив битов | O(1) | O(1) |

**Временная сложность для алгоритма, использующего массив символов или список** — константная, т.к все операции над множествами можно свести к двуместным операциям, следовательно сложность алгоритма зависит от временной сложности операции, имеющую наибольшую сложность. Для операции объединения над множествами нам нужно перебрать элементы одного и второго множества, т.е временная сложность O(n\*m), где n и m — мощности множеств, для которых происходит операция. Мощность множества не может превышать размера универсума. В худшем случае n = m = |U|, следовательно O(n\*m) = O(|U|\*|U|). Размер универсума в нашей задаче фиксирован (|U| = 26), значит временная сложность O(1).

**Временная сложность для алгоритма, использующего вектор битов** — константная, т.к операции сводятся к логическим операциям над битами множетва. Следовательно, временная сложность будет O(|U|). А размер универсума в нашей задаче фиксирован (|U| = 26), значит временная сложность O(1).

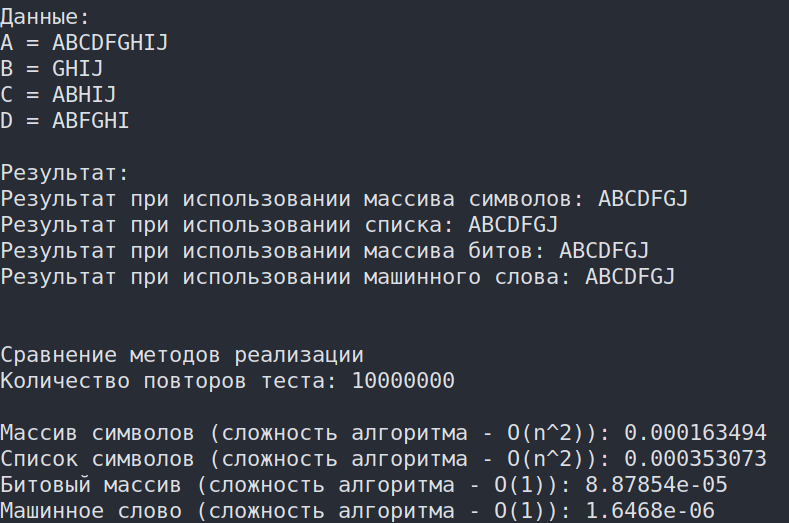
**Временная сложность для алгоритма, использующего машинное слово** — константная, т.к операции над множеством сводятся к битовым операциям над числами, которые выполняются за один шаг алгоритма, независимо от мощности множеств.

## **5. Результаты измерения времени обработки**

Таблица. 3. Результаты измерения времени обработки

|  |  |  |  |
| --- | --- | --- | --- |
| **Способ представления** | **Время (мс)** | **Количество повторов цикла** | **Зависимость от количества в множестве** |
| Массив символов | 0.0001642 - 0.0022 | 1000000 | есть |
| Список | 0.0003535 -  0.00322 | есть |
| Битовый массив | 0.000087 -  0.000108 | нет |
| Машинное слово | 0.0000016 -  0.000002 | нет |

Рисунок 1. Пример выполнения программы

****

## **6. Результаты решения задачи**

**Массив символов**

**Плюсы:**

1) Можно использовать при любой мощности универсума (если мощность заранее известна)

2) Удобность доступа к элементам, по сравнению с машинным словом.

**Минусы:**

1) Так как заранее не известна мощность множества, которая будет получена после операции, приходится выделять память с запасом (например, на весь универсум). Если памяти будет недостаточно, то придется выделить новый непрерывный участок памяти, который будет больше первоначального. Если мощность слишком большая, то такая опереация может быть невозможна.

**Линейный односвязный список**

**Плюсы:**

1) Можно использовать при любой мощности множества, даже если она заранее не известна

2) Память выделяется для каждого элемента множества, ровно столько, сколько необходимо.

**Минусы:**

1) С каждым элементом множества нужно хранить указатель на следующий элемент.

**Массив битов**

**Плюсы:**

1) Операции над множествами сводятся к логическим операциям над битами, не нужно реализовавать алгоритмы операции.

2) Можно использовать при любой мощности универсума, в отличие от машинного слова. (если мощность заранее известна)

**Минусы:**

1) Требует простой функции отображения множества на универсум. Если такая функция отсутствует, то данный метод хранения не применим для задачи.

**Машинное слово**

**Плюсы:**

1) При помощи маски можно обращаться сразу к нескольким элементам множества.

2) Операции над множествами сводятся к битовым операциям над числами, не нужно реализовавать алгоритмы операции.

**Минусы:**

1) Требует простой функции отображения множества на универсум. Если такая функция отсутствует, то данный метод хранения не применим для задачи.

2) Допустимо использовать метод только при размере универсума меньше размера машинного слова.

## Вывод

При выполнении задания были исследованы четыре способа представления множеств в памяти ЭВМ, оценена времененная сложность для алгоритма решения задачи для каждого из способа. Самым эффективным способом хранения в памяти ЭВМ множества для решения поставленной задачи является машинное слово.

## Приложение 1 (Листинги программы)

/\*

Яловега Никита 9308

Вариант 26

Задача:

U = Прописные латинские буквы

E = A/(B&C&D)

\*/

#include <iostream>

#include <cstdlib>

#include <ctime>

#include <string>

using namespace std;

// ----- Константы -----

const int U = 26; // мощность универсума

const int ROLLS = 10000000; // количество повторений теста

// ----- Структуры -----

struct List

// Структура односвязного линейного списка

{

char element;

List\* next;

List(char e, List\* n = nullptr): element(e), next(n){}

~List(){delete next;}

};

// ----- Прототипы функций -----

// функции конвертирования

char\* init\_arr(string);

char\* to\_array(List\*);

char\* to\_array(unsigned int);

char\* to\_array(bool\*);

unsigned int to\_word(char\*);

List\* to\_list(unsigned int);

bool\* to\_bool(unsigned int);

// функции реализации

char\* char\_array\_method(clock\_t\*, char\*, char\*, char\*, char\*);

List\* list\_method(clock\_t\*, List\*, List\*, List\*, List\*);

bool\* bool\_method(clock\_t\*, bool\*, bool\*, bool\*, bool\*);

unsigned int word\_method(clock\_t\*, unsigned int, unsigned int, unsigned int, unsigned int);

// декоративные фунции

void print\_table(double, double, double, double);

int main()

{

short menu;

clock\_t clock;

double time1, time2, time3, time4;

string buff;

char\* A = nullptr;

char\* B = nullptr;

char\* C = nullptr;

char\* D = nullptr;

char\* R = nullptr;

List\* lA = nullptr;

List\* lB = nullptr;

List\* lC = nullptr;

List\* lD = nullptr;

List\* lR = nullptr;

bool\* bA = nullptr;

bool\* bB = nullptr;

bool\* bC = nullptr;

bool\* bD = nullptr;

bool\* bR = nullptr;

unsigned int wA, wB, wC, wD, wR;

cout << "0) Подготовленный тест \n1) Ввод данных \n2) Генерация данных \n" << ">" << endl;

cin >> menu;

switch (menu)

{

case 1: // ввод данных

// Считываем строки, т.к самый удобный способ

cout << "Введите множество A: ";

cin >> buff;

A = init\_arr(buff);

wA = to\_word(A);

delete A;

cout << "Введите множество B: ";

cin >> buff;

B = init\_arr(buff);

wB = to\_word(B);

delete B;

cout << "Введите множество C: ";

cin >> buff;

C = init\_arr(buff);

wC = to\_word(C);

delete C;

cout << "Введите множество D: ";

cin >> buff;

D = init\_arr(buff);

wD = to\_word(D);

delete D;

break;

case 2: // генерация данных

// Генерируем машинные слова, т.к это проще всего

srand(time(nullptr));

wA = rand();

wB = rand();

wC = rand();

wD = rand();

break;

default: // Подготовленный тест

/\*

A = {'A', 'B', 'C', 'D', 'F', 'G', 'H', 'I', 'J', '\0'};

B = {'G', 'H', 'I', 'J', '\0'};

C = {'A', 'B', 'H', 'I', 'J', '\0'};

D = {'A', 'B', 'F', 'G', 'H', 'I', '\0'};

---

R = {'A', 'B', 'C', 'D', 'F', 'G', 'J', '\0'};

\*/

wA = 1007;

wB = 960;

wC = 899;

wD = 483;

}

// Основная функция

A = to\_array(wA);

B = to\_array(wB);

C = to\_array(wC);

D = to\_array(wD);

cout << "Данные: \n";

cout << "A = " << A << endl;

cout << "B = " << B << endl;

cout << "C = " << C << endl;

cout << "D = " << D << endl;

// Список

lA = to\_list(wA);

lB = to\_list(wB);

lC = to\_list(wC);

lD = to\_list(wD);

lR = list\_method(&clock, lA, lB, lC, lD);

time2 = 1000\*((((double)clock)/CLOCKS\_PER\_SEC)/ROLLS);

delete lA;

delete lB;

delete lC;

delete lD;

// Битовая цепочка

bA = to\_bool(wA);

bB = to\_bool(wB);

bC = to\_bool(wC);

bD = to\_bool(wD);

bR = bool\_method(&clock, bA, bB, bC, bD);

time3 = 1000\*((((double)clock)/CLOCKS\_PER\_SEC)/ROLLS);

delete bA;

delete bB;

delete bC;

delete bD;

// Машинное слово

wR = word\_method(&clock, wA, wB, wC, wD);

time4 = 1000\*((((double)clock)/CLOCKS\_PER\_SEC)/ROLLS);

// Массив символов

R = char\_array\_method(&clock, A, B, C, D);

time1 = 1000\*((((double)clock)/CLOCKS\_PER\_SEC)/ROLLS);

delete A;

delete B;

delete C;

delete D;

cout << "\nРезультат: \n";

cout << "Результат при использовании массива символов: " << R << endl;

delete R;

R = to\_array(lR);

cout << "Результат при использовании списка: " << R << endl;

delete R;

R = to\_array(bR);

cout << "Результат при использовании массива битов: " << R << endl;

delete R;

R = to\_array(wR);

cout << "Результат при использовании машинного слова: " << R << endl;

delete R;

delete bR;

delete lR;

print\_table(time1, time2, time3, time4);

return 0;

}

char\* char\_array\_method(clock\_t\* time, char\* A, char\* B, char\* C, char\* D)

/\*

Реализация задания при помощи структуры массива символов.

Сложность алгоритма O(n^2)

\*/

{

char\* E = new char[U+1];

char\* R = new char[U+1];

int rolls = ROLLS;

clock\_t start, stop;

int i, j, k, r, f;

start = clock();

for(;rolls > 0; rolls--)

{

// вычисление E = (B&C&D). Сложность - O(n^2)

for (i = 0, k = 0; B[i]; ++i)

for(j = 0; C[j]; ++j)

if (B[i] == C[j])

for (r = 0; D[r]; ++r)

if (B[i] == D[r])

E[k++] = B[i];

E[k] = '\0';

// Вычисление R = A / E. Сложность - O(n^2)

for (i = 0, k = 0; A[i]; ++i)

{

for (j = 0, f = 1; E[j] && f; ++j)

if (A[i] == E[j])

f = 0;

if (f == 1)

R[k++] = A[i];

}

R[k] = '\0';

}

stop = clock();

delete E;

\*time = stop-start;

return R;

}

List\* list\_method(clock\_t\* time, List\* A, List\* B, List\* C, List\* D)

/\*

Реализация задания при помощи структуры линейного односвязного списка.

Сложность алгоритма O(n^2)

\*/

{

List\* E = nullptr;

List\* R = nullptr;

int rolls = ROLLS;

clock\_t start, stop;

int f;

start = clock();

for(;rolls > 0; rolls--)

{

// вычисление E = (B&C&D). Сложность - O(n^2)

delete E;

E = nullptr;

for (auto x = B; x; x = x->next)

for(auto y = C; y; y = y->next)

if (x->element == y->element)

for (auto z = D; z; z = z->next)

if (x->element == z->element)

E = new List(x->element, E);

// Вычисление R = A / E. Сложность - O(n^2)

delete R;

R = nullptr;

for (auto x = A; x; x = x->next)

{

f = 1;

for (auto y = E; y && f; y = y->next)

if (x->element == y->element)

f = 0;

if (f == 1)

R = new List(x->element, R);

}

}

stop = clock();

delete E;

\*time = stop-start;

return R;

}

bool\* bool\_method(clock\_t\* time, bool\* A, bool\* B, bool\* C, bool\* D)

/\*

Реализация задания при помощи битовой цепочки.

Сложность алгоритма O(1)

\*/

{

bool\* result;

int rolls = ROLLS;

clock\_t start, stop;

result = new bool[U];

start = clock();

for(;rolls > 0; rolls--)

// Сложность O(1)

for(int i = 0; i < U; ++i)

result[i] = A[i] && !(B[i] && C[i] && D[i]);

stop = clock();

\*time = stop-start;

return result;

}

unsigned int word\_method(clock\_t\* time, unsigned int A, unsigned int B, unsigned int C, unsigned int D)

/\*

Реализация задания при помощи битовой цепочки.

Сложность алгоритма O(1)

\*/

{

unsigned int result;

int rolls = ROLLS;

clock\_t start, stop;

start = clock();

for(;rolls > 0; rolls--)

// Сложность O(1)

result = A & (~(B & C & D));

stop = clock();

\*time = stop-start;

return result;

}

char\* to\_array(unsigned int num)

{

char\* result = new char[U+1];

int i,j;

for(i = 0, j = 0; i < U; ++i)

if(1 << i & num)

result[j++] = i + 'A';

result[j] = '\0';

return result;

}

char\* to\_array(List\* set)

{

char\* result = new char[U+1];

int i,j;

for(j = 0; set; set = set->next)

result[j++] = set->element;

result[j] = '\0';

return result;

}

char\* to\_array(bool\* array)

{

char\* result = new char[U+1];

int i,j;

for(i = 0, j = 0; i < U; ++i)

if(array[i])

result[j++] = i + 'A';

result[j] = '\0';

return result;

}

List\* to\_list(unsigned int num)

{

List \*head = nullptr;

head = nullptr;

for (int i = 0; i < U; ++i)

if(1 << i & num)

head = new List(i + 'A', head);

return head;

}

unsigned int to\_word(char \*array)

{

unsigned int result = 0;

char c;

for(int i = 0; array[i]; ++i)

result |= 1 << (array[i] - 'A');

return result;

}

bool\* to\_bool(unsigned int num)

{

bool\* result = new bool[U];

int i = 0;

for(i = 0; i < U; ++i)

if(1 << i & num)

result[i] = true;

else

result[i] = false;

return result;

}

char\* init\_arr(string S) {

int i,j,n;

char\* ans;

n = S.length();

ans = new char[n+1];

for(i = 0; i < n; ++i)

ans[i] = S[i];

ans[i] = '\0';

return ans;

}

void print\_table(double time1, double time2, double time3, double time4)

/\*

Вывод таблицы на экран

\*/

{

cout << "\n\nCравнение методов реализации \nКоличество повторов теста: " << ROLLS << "\n\n";

cout << "Массив символов (сложность алгоритма - O(n^2)): " << time1 << endl;

cout << "Список символов (сложность алгоритма - O(n^2)): " << time2 << endl;

cout << "Битовый массив (сложность алгоритма - O(1)): " << time3 << endl;

cout << "Машинное слово (сложность алгоритма - O(1)): " << time4 << endl;

}