

МИНОБРАЗОВАНИЯ РОССИИ  
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ  
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ  
«ЛЭТИ» им.В.И.УЛЬЯНОВА (ЛЕНИНА)

Кафедра вычислительной техники

Отчет по лабораторной работе № 1  
по дисциплине «Алгоритмы и структуры данных, часть 1»  
Тема: «Множества»

Студенты гр. 9306

Евдокимов О.В. Кныш С.А. Павельев М.С.

Преподаватель

Манерагена Валентина

Санкт-Петербург  
2020

## Содержание

Цель.....	3
Задание.....	3
Контрольные тесты.....	3
Временная сложность.....	4
Результаты измерения времени.....	6
Выводы:.....	6
Список:.....	6
Машинное слово:.....	7
Массив битов:.....	7
Массив:.....	7
Код программы.....	7

## Цель

Исследование четырёх способов хранения множеств в памяти ЭВМ

## Задание

Универсум: Строчные латинские буквы

Что надо вычислить: Множество, содержащее все буквы, общие для множеств  $A$  и  $B$ , за исключением букв, содержащихся в  $C$ , а также все буквы из  $D$

Формализация задания, формула для вычисления пятого множества по четырём заданным —  $A \cap B \setminus C \cup D$

## Контрольные тесты

При запуске программы сначала нам выводятся тесты, в каждой строке конкретного теста находятся множества по порядку  $A, B, C, D$ . Далее пользователю требуется выбрать желаемый метод выполнения обработки множеств

```
Tests:

1 :
oezqipywajsbgmd
wbeh
mjb
sifemkbwh
-----

2 :
rnvziqpqa
fitnsb
sitaycx
zqkrevh
-----

3 :
vbzjrckap
qdtujkvfhaoe
nmybfhqespcrk
fwq
-----

Menu:
0: Exit
1: Process by word
2: Process by bool array
3: Process by list
4: Process by array

Enter your choice:
```

После выбора пользователю выводится результат обработки для каждого теста. После нажатия любой клавиши пользователю снова предоставляется выбор метода выполнения обработки для тех же тестов и так пока пользователь не выберет пункт в меню с номером 0 запускающий процесс выхода из программы.

```
Tests:

1 :
oezqipuwajsbgmd
wbeh
mjb
sifemkbwh
-----

2 :
rnvzjipqa
fitnsb
sitaycx
zqkrevh
-----

3 :
vbzjrckap
qdtujkvfhaoe
nmybfhqespcrk
fwq
-----

Menu:
0: Exit
1: Process by word
2: Process by bool array
3: Process by list
4: Process by array

Enter your choice: 1

answer for test 1: bef hikmsw
answer for test 2: ehknqrvz
answer for test 3: afjqvw

By word : 0.000333333
Для продолжения нажмите любую клавишу . . . █
```

## Временная сложность

Временная сложность перевода из строки в нужное представление.

N – размер универсума

Список:

$O(N) \Rightarrow$  константа

Машинное слово:

$O(N) \Rightarrow$  константа

Массив битов:

$O(N) \Rightarrow$  константа

Массив:

$O(0)$  (нет преобразования)  $\Rightarrow$  константа

Временная сложность операции пересечения:

$N$  – размер универсума

Список:

$O(N^2) \Rightarrow$  константа

Машинное слово:

$O(N) \Rightarrow$  константа

Массив битов:

$O(N^2) \Rightarrow$  константа

Массив:

В общем случае  $O(N^2) \Rightarrow$  константа

Временная сложность операции вычитания:

$N$  – размер универсума

Список:

В общем случае  $O(N^2) \Rightarrow$  константа

Машинное слово:

$O(N) \Rightarrow$  константа

Массив битов:

$O(N) \Rightarrow$  константа

Массив:

в общем случае  $O(N^3) \Rightarrow$  константа

Временная сложность операции объединения:

$N$  – размер универсума

Список:

В общем случае  $O(N^2) \Rightarrow$  константа

Машинное слово:

$O(N) \Rightarrow$  константа

Массив битов:

$O(N) \Rightarrow$  константа

Массив:

в общем случае  $O(N^2) \Rightarrow$  константа

Временная сложность операции перевода обратно в строку:

$N$  – размер универсума

Список:

В общем случае  $O(N^2) \Rightarrow$  константа

Машинное слово:

$O(N) \Rightarrow$  константа

Массив битов:

$O(N) \Rightarrow$  константа

Массив:

в общем случае  $O(N^2) \Rightarrow$  константа

## Результаты измерения времени

Способ представления / максимальный размер множества	26	13	6
Список	1.6826 e-05	1.235e-05	8.325e-06
Машинное слово	4.266e-06	4.242e-06	4.223e-06
Массив битов	6.798e-06	6.705e-06	6.485e-06
Массив	1.673e-06	1.046e-06	6.18e-07

## Выводы:

### Список:

Наиболее трудно реализуемый тип представления, также не отличающийся особой производительностью (скоростью выполнения). Может быть применен в тех случаях когда размер универсума никак не может быть предопределен.

## Машинное слово:

Наиболее простой в реализации тип представления. Может быть использован, в сфере где важна стабильность скорости вычислений, так как данный тип представления почти не зависит от размера входных данных

## Массив битов:

Имеет тоже достоинство, что и машинное слово, однако реализуется сложнее.

## Массив:

Наиболее быстрый тип представления. Может быть использован в сфере где важна скорость выполнения. Не желательно использовать при решении задач, размер универсума в которых не может быть предопределен.

## Код программы

```
#include <iostream>
#include <time.h>
#include <cstring>
using namespace std;

struct list {
    char el;
    list* next;
    list(char e, list* n = nullptr) : el(e), next(n) { }
    ~list() { delete next; }
};

list* delete_elem(list* lst, list* root);
void add(list** head, char c);
char* list_to_str(list* head);
int len(list* head);
void str_to_list(list* head, string in);
void and_list(list* a, list* b, list* e);
void sub_list(list* e, list* c);
void unite_list(list* e, list* d);
long int str_to_univers_word(string in);
char* univers_to_str_word(long int in);
char* univers_to_str_array(bool* in);
bool* str_to_univers_array(string in);
void and_bool_array(bool* a, bool* b, bool* e);
void initialize_bool(bool* a);
void sub_bool_array(bool* e, bool* c);
void unite_bool_array(bool* e, bool* d);
void del_elem_array(char* in, int n);
void unite_array(char* in_1, char* in_2);
void and_array(char* in1, char* in2, char* res);
void sub_array(char* in1, char* in2);
void proc_by_list(char** in, bool debug, int);
void proc_by_word(char** in, bool debug, int i);
void proc_by_bool_array(char** in, bool debug, int i);
void proc_by_array(char** in, bool debug, int);
```

```

char* generate_union();
void sort_array(char* in);

int main()
{
    clock_t start, stop;
    // A&&B\\C||D
    srand(time(0));

    const int N = 1000000;
    bool debug = false;
    int user_input = 1;

    char*** in = new char** [N];

    for (int j = 0; j < N; j++) {
        in[j] = new char* [4];
        for (int i = 0; i < 4; i++)
        {
            in[j][i] = generate_union();
        }
    }

    while (user_input != 0) {
        if (debug)
            printf("Tests:\n\n");
        for (int i = 0; i < N; i++)
        {
            if (debug)
                cout << i + 1 << " : " << endl;
            for (int j = 0; j < 4; j++)
            {
                if (debug)
                    cout << in[i][j] << endl;
            }
            if (debug)
                cout << "-----\n";
        }

        printf("\nMenu:\n0: Exit\n1: Process by word\n2: Process by
bool array\n3: Process by list\n4: Process by array\n\nEnter your
choice: ");

        cin >> user_input;
        cout << endl;
        switch (user_input)
        {
            case 1:
            {
                start = clock();
                for (int i = 0; i < N; i++) {
                    proc_by_word(in[i], debug, i + 1);

```



```

    }
    stop = clock();
    cout << "\nBy word : " << ((float)(stop - start) /
CLK_TCK) / N << endl;

    system("pause");
    user_input = 8;
    system("cls");
}
break;
case 2:
{
    start = clock();
    for (int i = 0; i < N; i++) {
        proc_by_bool_array(in[i], debug, i + 1);
    }
    stop = clock();
    cout << "\nBy bool array : " << ((float)(stop - start)
/ CLK_TCK) / N << endl;

    system("pause");
    user_input = 8;
    system("cls");
}
break;
case 3:
{
    start = clock();
    for (int i = 0; i < N; i++) {
        proc_by_list(in[i], debug, i + 1);
    }
    stop = clock();
    cout << "\nBy list : " << ((float)(stop - start) /
CLK_TCK) / N << endl;

    system("pause");
    user_input = 8;
    system("cls");
}
break;
case 4:
{
    start = clock();
    for (int i = 0; i < N; i++) {
        proc_by_array(in[i], debug, i + 1);
    }
    stop = clock();
    cout << "\nBy array : " << ((float)(stop - start) /
CLK_TCK) / N << endl;

    system("pause");
    user_input = 8;
    system("cls");
}

```

```

    }
    break;
    case 0:
    {
        for (int i = 0; i < N; i++) {
            for (int j = 0; j < 4; j++)
            {
                delete[] in[i][j];
            }
            delete[] in[i];
        }
        delete[] in;
    }break;
    default:
        break;
    }

}

system("pause");
return 0;
}

char* generate_union() {
    int m = rand() % 6;
    char* S = new char[26];
    char* St = new char[26];
    for (int i = 0; i < m; i++)
        S[i] = 49 + rand() % 0x1A + '0';
    int l = 0;
    for (int i = 0; i < m; i++) {
        bool flag = true;
        for (int j = 0; j < m; j++)
            if (i - j != 0 && S[i] == S[j]) flag = false;
        if (flag) {
            St[l++] = S[i];
        }
    }
    St[l] = '\0';
    delete[] S;
    return St;
}

void proc_by_bool_array(char** in, bool debug, int i)
{
    bool* a;
    bool* b;
    bool* c;
    bool* d;
    bool* e;

    a = str_to_univers_array(in[0]);

```

```

    b = str_to_univers_array(in[1]);

    c = str_to_univers_array(in[2]);

    d = str_to_univers_array(in[3]);

    e = str_to_univers_array("");

    and_bool_array(a, b, e);

    sub_bool_array(e, c);

    unite_bool_array(e, d);

    char* res;
    res = univers_to_str_array(e);
    if (debug)
        cout << "answer for test " << i << ": " << res << endl;

    delete[] res;
    delete[] a;
    delete[] b;
    delete[] c;
    delete[] d;
    delete[] e;
}

void proc_by_word(char** in, bool debug, int i)
{
    long int a;
    long int b;
    long int c;
    long int d;
    long int e;

    a = str_to_univers_word(in[0]);

    b = str_to_univers_word(in[1]);

    c = str_to_univers_word(in[2]);

    d = str_to_univers_word(in[3]);

    e = a & b;
    e = (e | c) & (~c);
    e = e | d;

    char* res;

    res = univers_to_str_word(e);
    if (debug)

```

```

        cout << "answer for test " << i << ": " << res << endl;

    delete[] res;
}

void proc_by_list(char** in, bool debug, int i)
{
    list* a = new list('#');
    list* b = new list('#');
    list* c = new list('#');
    list* d = new list('#');

    list* e = new list('#');

    str_to_list(a, in[0]);

    str_to_list(b, in[1]);

    str_to_list(c, in[2]);

    str_to_list(d, in[3]);

    and_list(a, b, e);

    sub_list(e, c);

    unite_list(e, d);

    char* res = list_to_str(e);
    if (debug)
        cout << "answer for test" << i << ": " << res << endl;

    delete[] res;
    delete a;
    delete b;
    delete c;
    delete d;
    delete e;
}

void proc_by_array(char** in, bool debug, int i)
{
    char* e = new char[26]();

    and_array(in[0], in[1], e);

    sub_array(e, in[2]);

    unite_array(e, in[3]);

    sort_array(e);
}

```

```

        if (debug)
            cout << "answer for test" << i << ": " << e << endl;

        delete[] e;
    }

void sort_array(char* in)
{
    char t[26];
    char cur;
    int l = 0;
    int i;
    int len_in = strlen(in);
    for (i = 0; i < 26; i++)
    {
        cur = 'a' + i;
        for (int j = 0; j < len_in; j++)
        {
            if (in[j] == cur)
                t[l++] = cur;
        }
    }
    t[l] = '\0';
    l = 0;
    for (i = 0; i < strlen(t); i++)
    {
        in[i] = t[i];
    }
    in[i] = '\0';
}

void unite_bool_array(bool* e, bool* d)
{
    for (int i = 0; i < 26; i++)
        e[i] = e[i] || d[i];
}

void sub_bool_array(bool* e, bool* c)
{
    for (int i = 0; i < 26; i++)
        e[i] = ((e[i] || c[i]) && !c[i]);
}

void initialize_bool(bool* a)
{
    for (int i = 0; i < 26; i++)
        a[i] = false;
}

void and_bool_array(bool* a, bool* b, bool* e)

```

```

{
    for (int i = 0; i < 26; i++)
        e[i] = bool(a[i] && b[i]);
}

char* univers_to_str_array(bool* in)
{
    char* k = new char[26];
    int m = 0;

    for (int i = 0; i < 26; i++)
        if (in[i])
            k[m++] = 'a' + i;

    k[m] = '\0';
    return k;
}

bool* str_to_univers_array(string in)
{
    bool* k = new bool[26];

    initialize_bool(k);

    for (char c : in)
        if (c != 0)
            k[int(c) - 97] = true;

    return k;
}

char* univers_to_str_word(long int in)
{
    char* k = new char[26];

    int m = 0;
    for (int i = 0; i < 26; i++) {
        if (in % 2 == 1)
            k[m++] = 'a' + i;
        in = in / 2;
    }

    k[m] = '\0';
    return k;
}

long int str_to_univers_word(string in)
{
    long int k = 0;;

    for (char c : in)
        if (c != 0)

```

```

        k = k | (1 << (int(c) - 97));

    return k;
}

void unite_list(list* e, list* d)
{
    list* curE = e->next;
    list* curD = d->next;
    bool flag = false;

    while (curD != nullptr)
    {
        curE = e->next;
        flag = false;
        while (curE != nullptr)
        {
            if (curE->el == curD->el)
                flag = true;

            curE = curE->next;
        }
        if (!flag)
            add(&e, curD->el);
        curD = curD->next;
    }
}

void and_list(list* a, list* b, list* e)
{
    list* curA = a->next;
    list* curB = b->next;

    while (curA != nullptr)
    {
        list* curB = b->next;
        while (curB != nullptr)
        {
            if (curA->el == curB->el)
                add(&e, curA->el);

            curB = curB->next;
        }
        curA = curA->next;
    }
}

void sub_list(list* e, list* c)
{
    list* curE = e->next;
    list* curC = c->next;

```

```

while (curE != nullptr)
{
    curC = c->next;
    while (curC != nullptr)
    {
        if (curE->el == curC->el)
        {
            curE = delete_elem(curE, e);
        }
        curC = curC->next;
    }
    curE = curE->next;
}

}

int len(list* head)
{
    list* cur;
    int c = 0;
    cur = head;
    while (cur->next != nullptr)
    {
        cur = cur->next;
        c++;
    }

    return c;
}

void str_to_list(list* head, string in)
{
    for (int i = 0; i < in.length(); i++)
        add(&head, in[i]);
}

char* list_to_str(list* head)
{
    list* a;
    char* res = new char[26];
    int l = 0;
    char c;

    for (int i = 0; i < 26; i++)
    {
        a = head->next;
        c = 'a' + i;
        while (a != nullptr)
        {
            if (a->el == c)
                res[l++] = a->el;
        }
    }
}

```



```

        a = a->next;
    }
}
res[1] = '\0';
return res;
}

void add(list** head, char c)
{
    list* a;
    list* wen;
    a = *head;

    try
    {
        wen = new list(c, a->next);
        a->next = wen;
    }
    catch (errc)
    {
        list* b = new list(c);
        b = new list('#', b);
        *head = b;
    }
}

list* delete_elem(list* lst, list* root)
{
    list* temp = nullptr;
    if (len(root) != 0) {
        temp = root;
        while (temp->next != lst)
        {
            temp = temp->next;
        }
        temp->next = lst->next;
        free(lst);
    }
    return(temp);
}

void del_elem_array(char* in, int n)
{
    int j = n;
    for (int i = n + 1; i < strlen(in); ++i) in[j++] = in[i];
    in[strlen(in) - 1] = '\0';
}

void unite_array(char* in1, char* in2)
{

```

```

    int f;
    for (int i = 0; i < strlen(in2); ++i)
    {
        f = 0;
        for (int j = 0; j < strlen(in1); ++j)
        {
            if (in2[i] == in1[j]) f = 1;
        }
        if (f == 0) in1[strlen(in1)] = in2[i];
    }
}

void and_array(char* in1, char* in2, char* res)
{
    int k = 0;
    for (int i = 0; i < strlen(in1); ++i)
    {
        for (int j = 0; j < strlen(in2); ++j)
        {
            if (in1[i] == in2[j]) res[k++] = in1[i];
        }
    }
}

void sub_array(char* in1, char* in2)
{
    for (int i = 0; i < strlen(in1); ++i)
    {
        for (int j = 0; j < strlen(in2); ++j)
        {
            if (in1[i] == in2[j]) del_elem_array(in1, i);
        }
    }
}

```