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 34**

|  |  |
| --- | --- |
| Выполнили студенты гр. 9308: | Дементьев Д.П. Кашапова О.С. |
| 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 5](#__RefHeading___Toc5004_2043932214)

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

[​ Вывод 7](#__RefHeading___Toc5008_2043932214)

[​ Список используемых источников 8](#__RefHeading___Toc439_358905059)

[​ Приложение 1 (Листинги программы) 9](#__RefHeading___Toc441_358905059)

## Введение

Целью задания является исследование четырех способов хранения множеств в памяти ЭВМ

## **1. Задание**

Множество содержащие все буквы из A, не содержащиеся в B, C и D.

Универсум — строчные латинские буквы

## 2. Формализация задания

Условие задачи можно записать в виде формулы: E = ((A\B)\C)\D

## **3. Контрольные примеры**

Контрольные примеры представлены в таблице 1*.*

Таблица. 1. Контрольные примеры

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Исходные множества** | | | | **Результат** |
| **A** | **B** | **C** | **D** | **E** |
| 1 | ABCDFGHIJ | GHIJ | ABHIJ | ABFGHI | CD |
| 2 | WYXZ | HJYW | ABCDE | NAVI | YXZ |
| 3 | ABCFOITY | BF | IY | F | ACOT |
| 4 | ABC | DEF | GHI | XYZ | ABC |
| 5 | ABC | ABC | ABC | ABC | Пустое множество |

## **4. Временная сложность**

Временная сложность представлена в таблице 2*.*

Таблица. 2. Временная сложность

|  |  |  |
| --- | --- | --- |
| **Способ представления** | **Ожидаемая** | **Фактическая** |
| Последовательность | O(n^2) | O(n^2) |
| Список | O(n^2) | O(n^2) |
| Машинное слово | O(1) | O(1) |
| Массив битов | O(1) | O(1) |

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

Результаты измерения времени обработкипредставлены в таблице 3*.*

Таблица. 3. Результаты измерения времени обработки

|  |  |  |  |
| --- | --- | --- | --- |
| **Способ представления** | **Время(мс)** | **Количество повторов цикла** | **Зависимость от количества в множестве** |
| Массив символов | 0.000191 - 0.002 | 1000000 | есть |
| Список | 0.001186 - 0.005 | есть |
| Машинное слово | 2\*10-6 | нет |
| Массив битов | 3.6\*10-5 | нет |

## **6. Результаты решения задачи**

Ниже приведён пример работы программы, использующей различные способы решения задачи:

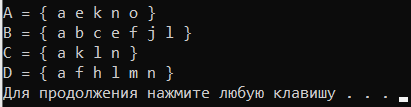


Рис.1. Исходные данные

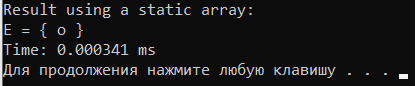


Рис.2. Массив символов

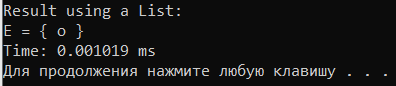


Рис.3. Линейный односвязный список

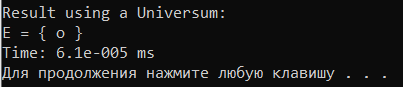


Рис.4. Массив битов

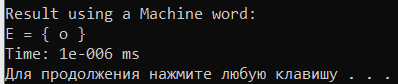


Рис.5. Машинное слово

## 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, но важно учитывать, что мощность универсума не должна превышать разрядность вычислительной машине. Подобным ограничением не обладают другие способы представления множеств в памяти ЭВМ.

## Список используемых источников

## Приложение 1 (Листинги программы)

#include <stdio.h>

#include <iostream>

#include <stdlib.h>

#include <time.h>

/\* Формула E = ((A\B)\C)\D

Контрольный тест:

A={'a','b','c','d'}

B={'b','e','f'}

C={'d','g','h'}

D={'b','d', 'c'}

Ожидаемый ответ:

E={'a'}

\*/

using namespace std;

const int N=26;

struct NODE

{

NODE \*next;

char el;

};

// Функции для работы со статичным массивом

void process(char A[], char B[], char E[])

{

int i=0, j=0, k=0;

bool flag;

while (A[i]!= 0) {

flag=true;

j=0;

while (B[j] != 0){

//cout << A[i] << ' ' << B[j] << endl;

if (A[i]==B[j]) flag=false;

j++;

}

if (flag) {

E[k]=A[i];

(k)++;

}

i++;

}

E[k]=0;

}

void print(char E[])

{

int i=0;

cout << "{ ";

while (E[i] != 0) printf("%c ", E[i++]);

printf("}\n");

}

// Функции для осуществления работы со списком

NODE \*make\_head()

{

NODE \*HEAD = NULL;

HEAD = (NODE\*)malloc(sizeof(NODE));

if (HEAD) HEAD->next=NULL;

return HEAD;

}

NODE \*create(char sym)

{

NODE \*node=NULL;

node = (NODE\*)malloc(sizeof(NODE));

if (node) {

node->el=sym;

node->next=NULL;

}

return node;

}

void add(NODE \*head, NODE \*node)

{

NODE \*p=head;

if (!(head->next)) head->next=node;

else {

while (p->next != NULL) p = p->next;

p->next = node;

}

}

void strToList(NODE \*HEAD, char A[])

{

int i=0;

NODE \*p=NULL;

while (A[i] != 0) {

p=create(A[i]);

add(HEAD, p);

i++;

}

}

NODE \*process(NODE \*HEAD\_A, NODE \*HEAD\_B)

{

bool flag;

NODE \*p\_a, \*p\_b, \*HEAD\_E, \*p\_e;

HEAD\_E = make\_head();

p\_a = HEAD\_A->next;

while (p\_a) {

flag=true;

p\_b=HEAD\_B->next;

while (p\_b){

//cout << p\_a->el << ' ' << p\_b->el << endl;

if (p\_a->el==p\_b->el) flag=false;

p\_b=p\_b->next;

}

if (flag) {

p\_e = create(p\_a->el);

add(HEAD\_E, p\_e);

}

p\_a=p\_a->next;

}

return HEAD\_E;

}

void print(NODE \*HEAD)

{

NODE \*p=HEAD;

cout << "{ ";

if (!(HEAD->next)) cout << " ";

else {

do {

p = p->next;

cout << p->el << " ";

} while (p->next);

}

cout << "}" << endl;

}

void clean(NODE \*head)

{

NODE \*p=head->next,

\*buff=nullptr;

if (p)

{

while (p)

{

buff = p;

p = p->next;

buff->next = nullptr;

delete buff;

}

}

}

// Функции для работы с массивом boolean'ов

void strToUniverse(char A[], bool U[])

{

int i=0;

while (A[i] != 0) {

U[(int)A[i]-'a']=1;

i++;

}

}

void process(bool U[], char A[])

{

int i=0;

while (A[i]!=0) {

U[(int)A[i]-'a']=0;

i++;

}

}

void print(bool U[])

{

int i;

cout << "{ ";

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

if (U[i]!=0) cout << (char)(i+'a') << " ";

cout << "}" << endl;

}

// Функции для работы с машинным словом

void strToWord(int \*word, char A[])

{

int i=0;

while (A[i] != 0) {

\*word = \*word | (1 << (int)(A[i]-'a'));

i++;

}

}

void print(int word)

{

int i;

cout << "{ ";

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

if (word & (1 << i)) cout << (char)(i+'a') << " ";

cout << "}" << endl;

}

// Общие функции

int Menu(int i)

{

char c;

int Q;

system("cls");

printf("What you want to do?\n");

switch (i)

{

case 0:

printf("1 - Input data\n");

printf("2 - Processing method\n");

printf("3 - Output initial data\n");

printf("0 - Exit\n");

break;

case 11:

printf("1 - Enter data\n");

printf("2 - Generate data\n");

printf("0 - Back\n");

break;

case 12:

printf("1 - Use static array\n");

printf("2 - Use a list\n");

printf("3 - Use array of boolean\n");

printf("4 - Use machine word\n");

printf("0 - Back\n");

break;

}

printf("Your choise - ");

scanf("%d", &Q);

while ((c=getchar())!='\n');

return Q;

}

char \*initArr(string S, int \*n) {

int i, j;

char \*ans; \*n = S.length();

ans = new char[\*n];

for (i = 0, j = 0; i < \*n; ++i) {

ans[j] = S[i];

j++;

}

return ans;

}

void generateData(char A[])

{

int gen, i, k=0;

gen = rand() % 0x3FFFFFF;

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

{

if (gen & 1<<i) A[k++] = (char)(i+'a');

}

A[k]='\0';

}

int main()

{

srand(time(NULL));

NODE \*HEAD\_A=NULL, \*HEAD\_B=NULL, \*HEAD\_C=NULL, \*HEAD\_D=NULL, \*HEAD\_E=NULL;

int wA=0, wB=0, wC=0, wD=0, wE=0, Q, Q11, Q12;

char A[N+1]={'a','b','c','d'};

char B[N+1]={'b','e','f'};

char C[N+1]={'d','g','h'};

char D[N+1]={'b','d'};

char E[N+1]={};

bool U[N] = {};

int r, rolls = 1000000;

clock\_t ticks, start;

char ans[N+1] = {};

int i;

do

{

Q = Menu(0);

switch (Q)

{

case 1:

do

{

Q11 = Menu(11);

switch (Q11)

{

case 1:

cout << "Input A: "; cin >> A;

cout << "Input B: "; cin >> B;

cout << "Input C: "; cin >> C;

cout << "Input D: "; cin >> D;

Q11=0;

break;

case 2:

cout << "Gegenerated A: ";

generateData(A);

print(A);

cout << "Gegenerated B: ";

generateData(B);

print(B);

cout << "Gegenerated C: ";

generateData(C);

print(C);

cout << "Gegenerated D: ";

generateData(D);

print(D);

system("pause");

Q11=0;

case 0:

break;

default:

printf("Error. Try again...\n");

}

} while (Q11 != 0);

break;

case 2:

do

{

Q12 = Menu(12);

switch (Q12)

{

case 1:

start = clock();

for (r=rolls;r>0;r--)

{

for (i=0; i<N; i++) ans[i]=E[i];

process(A, B, ans); // Преобразования статичными массивами

process(ans, C, ans);

process(ans, D, ans);

}

ticks = clock() - start;

cout << "Result using a static array:" << endl;

cout << "E = "; print(ans);

cout << "Time: " << 1000 \* ((((double) ticks) / CLOCKS\_PER\_SEC) / rolls) << " ms" << endl;

Q12=0;

break;

case 2:

if (HEAD\_A)

{

clean(HEAD\_A);

clean(HEAD\_B);

clean(HEAD\_C);

clean(HEAD\_D);

clean(HEAD\_E);

}

HEAD\_A = make\_head();

strToList(HEAD\_A, A); // Преобразование строки в список

HEAD\_B = make\_head();

strToList(HEAD\_B, B);

HEAD\_C = make\_head();

strToList(HEAD\_C, C);

HEAD\_D = make\_head();

strToList(HEAD\_D, D);

start = clock();

for (r=rolls; r>0; r--)

{

HEAD\_E = process(HEAD\_A, HEAD\_B); // Преобразования списком

HEAD\_E = process(HEAD\_E, HEAD\_C);

HEAD\_E = process(HEAD\_E, HEAD\_D);

}

ticks = clock()-start;

cout << "Result using a List:" << endl;

cout << "E = "; print(HEAD\_E); // Вывод в виде списка

cout << "Time: " << 1000 \* ((((double) ticks) / CLOCKS\_PER\_SEC) / rolls) << " ms" << endl;

Q12=0;

break;

case 3:

strToUniverse(A, U); // Преобразование строки в универсум

start = clock();

for (r=rolls; r>0; r--)

{

process(U, B); // Преобразования при помозщи универсума

process(U, C);

process(U, D);

}

ticks = clock() - start;

cout << "Result using a Universum:" << endl;

cout << "E = "; print(U); // Вывод по составленному универсуму

cout << "Time: " << 1000 \* ((((double) ticks) / CLOCKS\_PER\_SEC) / rolls) << " ms" << endl;

Q12=0;

break;

case 4:

strToWord(&wA, A); // Преобразование строки в машинное слово

strToWord(&wB, B);

strToWord(&wC, C);

strToWord(&wD, D);

start = clock();

for (r=rolls; r>0; r--)

wE = ((wA & ~wB) & ~wC) & ~wD; // Преобразование машинным слово

ticks = clock() - start;

cout << "Result using a Machine word:" << endl;

cout << "E = "; print(wE); // Вывод по машинному слову

cout << "Time: " << 1000 \* ((((double) ticks) / CLOCKS\_PER\_SEC) / rolls) << " ms" << endl;

Q12=0;

break;

case 0:

break;

default:

printf("Error. Try again...\n");

}

system("pause");

} while (Q12!=0);

break;

case 3:

cout << "A = "; print(A);

cout << "B = "; print(B);

cout << "C = "; print(C);

cout << "D = "; print(D);

system("pause");

break;

case 0:

printf("Exit...");

break;

default:

printf("Error. Try again...\n");

}

} while (Q !=0);

if (HEAD\_A)

{

clean(HEAD\_A);

clean(HEAD\_B);

clean(HEAD\_C);

clean(HEAD\_D);

clean(HEAD\_E);

}

return 0;

}