



Министерство науки и высшего образования Российской Федерации  
Калужский филиал  
федерального государственного бюджетного  
образовательного учреждения высшего образования  
«Московский государственный технический университет имени Н.Э. Баумана  
(национальный исследовательский университет)»  
(КФ МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИУК «Информатика и управление»  
КАФЕДРА ИУК4 «Программное обеспечение ЭВМ,  
информационные технологии»

## ЛАБОРАТОРНАЯ РАБОТА №4

### «Обработка разряженных матриц»

ДИСЦИПЛИНА: «Типы и структуры данных»

Выполнил: студент гр. ИУК4-31Б

  
(подпись)

( Суриков Н. С. )  
(Ф.И.О.)

Проверил:

  
(подпись)

( Былинка М. И. )  
(Ф.И.О.)

Дата сдачи (защиты): 20.12.24

Результаты сдачи (защиты):

- Балльная оценка: 85

- Оценка:

**Цель:** формирование практических навыков реализации хранения и обработки разреженных матриц.

**Задачи:**

1. Познакомиться с понятиями разреженная/плотная матрица.
2. Изучить алгоритмы представления матриц в компактной форме.
3. Научиться реализовывать изученные алгоритмы средствами ООП-технологии.
4. Реализовать операцию над разреженными матрицами согласно варианту.

**Вариант 4**

**Формулировка задания:**

*Разреженные матрицы  $(A)_{n \times m}$  и  $(B)_{n \times m}$  хранятся в профильной схеме.*

*Смоделировать операцию сложения двух матриц с получением результата в том же формате.*

**Листинг программы:**

**main.cpp**

```
1  #include <iostream>
2  #include <ctime>
3  #include <random>
4  #include "ProfileMatrix.h"
5  #include <sstream>
6  #include <iomanip>
7  #include <algorithm>
8
9  template <typename T>
10 std::ostream& operator<<(std::ostream& out, const std::vector<T>& v)
11 {
12     out << "[";
13     for (size_t i = 0; i < v.size(); i++)
14     {
15         out << v[i] << (i == v.size() - 1 ? "" : ", ");
16     }
17     out << "]";
18     return out;
19 }
20
21 template <typename T>
22 std::ostream& operator<<(std::ostream& out, const std::vector<std::vector<T>>&
vv)
```

```

23 {
24     int max_length = 0;
25     for (auto v : vv)
26     {
27         max_length = std::to_string(*std::max_element(v.begin(),
v.end())).length();
28     }
29     max_length += 2;
30     for (auto v : vv)
31     {
32         for (auto i : v)
33         {
34             std::cout << std::setw(max_length) << i;
35         }
36         std::cout << "\n";
37     }
38     return out;
39 }
40
41 inline int getRandom(const int _min, const int _max)
42 {
43     return rand() % (_max - _min + 1) + _min;
44 }
45
46 std::vector<std::vector<int>> getBandMatrix(const int N)
47 {
48     std::vector<std::vector<int>> matrix(N, std::vector<int>(N));
49     for (int i = 0; i < N; i++)
50     {
51         for (int j = 0; j < N; j++)
52         {
53             int bandwidth = getRandom(1, N / 2);
54             if (i >= j)
55             {
56                 matrix[i][j] = matrix[j][i] = (j <= i - bandwidth ? 0 : rand()
% 100);
57                 if (rand() % 5 == 0 && i != j)
58                 {
59                     matrix[i][j] = matrix[j][i] = 0;
60                 }
61             }
62         }
63     }
64     return matrix;
65 }
66
67 int main()
68 {
69     srand(time(nullptr));
70     ProfileMatrixStorageScheme::Matrix matrixA = getBandMatrix(10);
71     ProfileMatrixStorageScheme::Matrix matrixB = getBandMatrix(10);
72     ProfileMatrixStorageScheme pmssA(matrixA);
73     ProfileMatrixStorageScheme pmssB(matrixB);
74
75     std::cout << "matrix A = \n" << pmssA.getMatrix();

```

```

76     std::cout << "ANA = " << pmssA.getAN() << "\n";
77     std::cout << "IAA = " << pmssA.getIA() << "\n\n";
78
79     std::cout << "matrix B = \n" << pmssB.getMatrix();
80     std::cout << "ANB = " << pmssB.getAN() << "\n";
81     std::cout << "IAB = " << pmssB.getIA() << "\n\n";
82
83     ProfileMatrixStorageScheme pmssC = pmssA + pmssB;
84     std::cout << "matrix C = A + B =\n" << pmssC.getMatrix();
85     std::cout << "ANC = " << pmssC.getAN() << "\n";
86     std::cout << "IAC = " << pmssC.getIA() << "\n";
87
88     return 0;
89 }

```

## ProfileMatrix.cpp

```

1  #include "ProfileMatrix.h"
2
3  ProfileMatrixStorageScheme::ProfileMatrixStorageScheme(const Matrix& _matrix)
4  {
5      for (auto i : _matrix)
6      {
7          if (i.size() != _matrix.size())
8          {
9              throw std::invalid_argument("Matrix must be symmetrical");
10         }
11     }
12
13     const int N = _matrix.size();
14
15     for (size_t i = 0; i < N; i++)
16     {
17         for (size_t j = 0; j < N; j++)
18         {
19             if (_matrix[i][j] != _matrix[j][i])
20             {
21                 throw std::invalid_argument("Matrix must be symmetrical");
22             }
23         }
24     }
25
26     matrix = _matrix;
27
28     int cnt = 0;
29     for (int i = 0; i < N; i++)
30     {
31         bool begin = false;
32         for (int j = 0; j <= i; j++)
33         {
34             if (matrix[i][j] != 0)
35             {
36                 begin = true;
37             }

```

```

38         if (begin)
39         {
40             cnt++;
41             AN.push_back(matrix[i][j]);
42         }
43     }
44     IA.push_back(cnt);
45 }
46 }
47
48 ProfileMatrixStorageScheme::ProfileMatrixStorageScheme(const
std::vector<int>& _AN, const std::vector<int>& _IA)
49 {
50     AN = _AN;
51     IA = _IA;
52     const int N = IA.size();
53     matrix = std::vector<std::vector<int>>(N, std::vector<int>(N));
54     matrix[0][0] = AN[0];
55     for (int i = 1; i < N; i++)
56     {
57         for (int j = IA[i - 1]; j < IA[i]; j++)
58         {
59             int m = IA[i] - IA[i - 1];
60             int jmini = i - m + 1;
61             matrix[i][jmini + j - IA[i - 1]] = AN[j];
62         }
63     }
64     for (int i = 0; i < N; i++)
65     {
66         for (int j = 0; j < N; j++)
67         {
68             if (i > j)
69             {
70                 matrix[j][i] = matrix[i][j];
71             }
72         }
73     }
74 }
75
76 ProfileMatrixStorageScheme::Matrix ProfileMatrixStorageScheme::getMatrix()
const
77 {
78     return matrix;
79 }
80
81 std::vector<int> ProfileMatrixStorageScheme::getAN() const
82 {
83     return AN;
84 }
85
86 std::vector<int> ProfileMatrixStorageScheme::getIA() const
87 {
88     return IA;
89 }
90

```

```

91 ProfileMatrixStorageScheme operator+(const ProfileMatrixStorageScheme& pmssA,
const ProfileMatrixStorageScheme& pmssB)
92 {
93     auto A = pmssA.getMatrix();
94     auto B = pmssB.getMatrix();
95
96     auto ANA = pmssA.getAN();
97     auto IAA = pmssA.getIA();
98
99     auto ANB = pmssB.getAN();
100    auto IAB = pmssB.getIA();
101
102    if (A.size() != B.size())
103    {
104        throw std::invalid_argument("Matrices must be of the same order");
105    }
106
107    const int N = A.size();
108    std::vector<int> ANC;
109    std::vector<int> IAC;
110    ANC.push_back(ANA[0] + ANB[0]);
111    for (int i = 1; i < N; i++)
112    {
113        std::vector<int> striA;
114        std::vector<int> striB;
115
116        for (int j = IAA[i - 1]; j < IAA[i]; j++)
117        {
118            striA.push_back(ANA[j]);
119        }
120
121        for (int j = IAB[i - 1]; j < IAB[i]; j++)
122        {
123            striB.push_back(ANB[j]);
124        }
125
126        std::vector<int> mnstri = (striA.size() > striB.size() ? striB :
striA);
127        std::vector<int> mxstri = (striA.size() > striB.size() ? striA :
striB);
128
129        std::reverse(mnstri.begin(), mnstri.end());
130        std::reverse(mxstri.begin(), mxstri.end());
131
132        for (int j = 0; j < mnstri.size(); j++)
133        {
134            mxstri[j] += mnstri[j];
135        }
136        std::reverse(mxstri.begin(), mxstri.end());
137        for (auto j : mxstri)
138        {
139            ANC.push_back(j);
140        }
141        IAC.push_back(ANC.size());
142    }

```

```

143     IAC.insert(IAC.begin(), 1);
144     ProfileMatrixStorageScheme pmssC(ANC, IAC);
145     return pmssC;
146 }

```

## ProfileMatrix.h

```

1  #ifndef PMSS
2  #define PMSS
3  #include <vector>
4  #include <stdexcept>
5  #include <algorithm>
6
7  class ProfileMatrixStorageScheme
8  {
9  public:
10     using Matrix = std::vector<std::vector<int>>>;
11
12     ProfileMatrixStorageScheme(const Matrix& matrix);
13     ProfileMatrixStorageScheme(const std::vector<int>& AN, const
std::vector<int>& IA);
14     Matrix getMatrix() const;
15     std::vector<int> getAN() const;
16     std::vector<int> getIA() const;
17     friend ProfileMatrixStorageScheme operator+(const
ProfileMatrixStorageScheme& pmss1, const ProfileMatrixStorageScheme& pmss);
18
19 private:
20     Matrix matrix;
21     std::vector<int> AN;
22     std::vector<int> IA;
23 };
24 #endif

```

## Результат работы:

```

matrix A =
41 60 89 0 0 0 0 0 0 0
60 97 86 0 0 0 0 0 0 0
89 86 31 0 92 0 0 0 0 0
0 0 0 17 85 0 0 0 0 0
0 0 92 85 49 38 46 26 0 0
0 0 0 0 38 39 96 0 86 0
0 0 0 0 46 96 64 66 80 0
0 0 0 0 26 0 66 74 0 63
0 0 0 0 0 86 80 0 21 0
0 0 0 0 0 0 0 63 0 62
ANA = [41, 60, 97, 89, 86, 31, 17, 92, 85, 49, 38, 39, 46, 96, 64, 26, 0, 66, 74,
86, 80, 0, 21, 63, 0, 62]
IAA = [1, 3, 6, 7, 10, 12, 15, 19, 23, 26]

matrix B =
42 0 0 0 0 0 0 0 0 0

```

0	25	0	65	0	0	0	0	0	0
0	0	94	79	60	78	0	0	0	0
0	65	79	25	33	53	0	0	0	0
0	0	60	33	22	78	94	0	0	0
0	0	78	53	78	51	0	0	0	0
0	0	0	0	94	0	2	0	25	85
0	0	0	0	0	0	0	0	19	0
0	0	0	0	0	0	25	19	92	30
0	0	0	0	0	0	85	0	30	39

ANB = [42, 25, 94, 65, 79, 25, 60, 33, 22, 78, 53, 78, 51, 94, 0, 2, 25, 19, 92, 85, 0, 30, 39]  
 IAB = [1, 2, 3, 6, 9, 13, 16, 16, 19, 23]

matrix C = A + B =

83	60	89	0	0	0	0	0	0	0
60	122	86	65	0	0	0	0	0	0
89	86	125	79	152	78	0	0	0	0
0	65	79	42	118	53	0	0	0	0
0	0	152	118	71	116	140	26	0	0
0	0	78	53	116	90	96	0	86	0
0	0	0	0	140	96	66	66	105	85
0	0	0	0	26	0	66	74	19	63
0	0	0	0	0	86	105	19	113	30
0	0	0	0	0	0	85	63	30	101

ANC = [83, 60, 122, 89, 86, 125, 65, 79, 42, 152, 118, 71, 78, 53, 116, 90, 140, 96, 66, 26, 0, 66, 74, 86, 105, 19, 113, 85, 63, 30, 101]  
 IAC = [1, 3, 6, 9, 12, 16, 19, 23, 27, 31]

**Вывод:** в ходе работы были сформированы практические навыки создания, хранения и обработки разреженных матриц.