УДК 620.1.08

ПРЕДСТАВЛЕНИЕ РАЗРЕЖЕННЫХ МАТРИЦ С ИСПОЛЬЗОВАНИЕМ АССОЦИАТИВНЫХ КОНТЕЙНЕРОВ C++ БИБЛИОТЕКИ STL

канд. техн. наук, доц. Д.О. ГЛУХОВ, канд. техн. наук, доц. Р.П. БОГУШ, Т.М. ГЛУХОВА
(Полоцкий государственный университет)

Предлагается объектно-ориентированный архитектурный шаблон для построения ассоциативных контейнеров, предназначенных для представления разреженных матриц. Решается проблема классических ассоциативных контейнеров STL, состоящая в появлении несуществующих элементов при первом обращении к ним. Предлагаемый архитектурный шаблон не меняет вычислительной сложности основных операций ассоциативного контейнера std::map и обеспечивает компактное представление разреженной матрицы жесткости системы линейных уравнений большой размерности метода конечных элементов в задаче расчета статически неопределимых строительных конструкций с учетом физической и геометрической нелинейности.

Ключевые слова: *метод конечных элементов, разреженная матрица, объектно-ориентированные шаблоны проектирования, ассоциативный контейнер.*

Описание проблемы. При решении задач строительной механики методом конечных элементов из системы линейных алгебраических уравнений (СЛАУ) определяются узловые перемещения. Типовым представлением системы линейных уравнений является

$$[K]\{r\} = \{P\}, \tag{1}$$

где $[K]$ – матрица жесткости стержневой системы, формируемая как комбинация матриц жесткости отдельных конечных элементов;

$\{r\}$ – вектор узловых перемещений;

$\{P\}$ – вектор узловых нагрузок.

Моделируемые конструкции содержат десятки тысяч конечных элементов, что приводит к необходимости проработки программных решений для представления матрицы жесткости большой размерности в памяти компьютера. Важной особенностью представляемых матриц является их разреженность.

При переходе к нелинейным моделям строительных конструкций в процессе расчета возникает задача дополнения расчетной схемы более детальным конечно-элементным представлением для учета процессов трещинообразования (образования пластических шарниров и перераспределения внутренних усилий). Таким образом, топология расчетной схемы меняется в процессе расчета методом последовательных нагружений, порождая на каждом этапе расчета модифицированную матрицу жесткости (1), возможно, большего ранга, чем предшествующая.

Решение таких систем строится на прямых или итерационных методах с применением различных разложений матриц, повышающих обусловленность задачи, что существенно увеличивает объем матрично представленной информации. Преимущество итерационных методов и алгоритмов заключается в отсутствии необходимости строить и хранить в памяти компьютера матрицы большой размерности. Недостатками итерационных методов являются чувствительность к выбору первого приближения и плохая сходимость в плохо обусловленных задачах. Однако известны способы уменьшения погрешности решения (например, алгоритм итерационного уточнения [5–7]).

В языке программирования C++ для решения задачи построения динамических ассоциативных контейнеров в рамках объектно-ориентированного подхода широкое распространение получила стандартная библиотека шаблонов (Standard Template Library, STL) и ее модификации в рамках библиотеки boost. Контейнеры данной библиотеки обеспечивают оптимальную работу с памятью и известную вычислительную сложность операций вставки, удаления и поиска элементов.

В силу своей универсальности контейнеры STL способны инстанцироваться самими классами контейнеров, создавая многомерную структуру.

Рассмотрим пример 1:

```
typedef std::map<int, double> IDM;
IDM m;
m[10] = 23.12;
```

Ассоциативный контейнер std::map представляет собой отсортированный вектор пар «ключ – значение». Он предлагает удобные средства добавления, поиска и удаления элемента по ключу. Вычислительная сложность операций вставки, удаления и поиска равна $O(\log n)$. В своей реализации класс std::map использует деревья двоичного поиска.

Ключевым недостатком данного контейнера является то, что при обращении к несуществующему элементу данный элемент создается и размещается в памяти.

Рассмотрим пример 2:

```
typedef std::map<int, double> IDM;
IDM m;
m[10] = 23.12;
int s1 = m.size();          // вернет 1
double a = m[13];           // в переменную a будет помещен 0, так как элемент m[13] не заносился
double b = m[10];           // в переменную b будет помещено число 23.12
int s2 = m.size();          // вернет 2, хотя элемент m[13] в вектор не заносился
```

Причинами изменения размера контейнера при обращении к несуществующим элементам являются особенности реализации стандартных операторов присваивания operator=, определенных для типа помещаемых в контейнер значений, и оператора извлечения ссылки на объект operator[], определенного для соответствующего типа контейнера. Поскольку оператор извлечения ссылки operator[] срабатывает до оператора присваивания, то он «не знает» о том, что со ссылкой будет делать последующий код программы. В свою очередь оператор присваивания для типа (например, double) «не имеет никакого представления», как ссылка на объект была сформирована, была ли она получена в результате обращения к контейнеру. Такая последовательность событий требует обязательного создания несуществующего элемента ассоциативного контейнера.

Универсальность контейнера позволяет формировать и многомерные структуры. Покажем это на примере 3:

```
typedef std::map<int, double> IDM;
typedef std::map<int, IDM> IIDM;
IIDM m;
m[5][15] = 3.45;            m[5][3] = 2.55; m[3][17] = 32.7;
int s1 = m.size();          // вернет 2
int s2 = m[5].size();       // вернет 2
int s3 = m[3].size();       // вернет 1
int s4 = m[2].size();       // вернет 0
double a = m[2][2];         // в переменную a будет помещен 0, так как элемент m[2][2] не заносился
double b = m[3][17];        // в переменную b будет помещено число 32.7
s1 = m.size();              // вернет 3, хотя ожидалось 2, так как изменений в карте не было, а
s4 = m[2].size();           // вернет 1, хотя ожидался 0
```

Удобство работы с такими структурами при разработке численных алгоритмов поиска решений СЛАУ очевидно. Однако тот факт, что при обращении к ним наблюдается появление несуществующих элементов, делает данные контейнерные классы неприменимыми для задач большой размерности с разреженными матрицами, к которым относятся и задачи строительной механики.

Работы по поиску оптимальных структур для хранения и манипулирования разреженными матрицами публиковались в разное время рядом авторов (А.И. Богоявленский, Bathe Klaus-Jürgen, А.Б. Свириденко, О.А. Дмитриева, С. Писсанецки и др.). В частности, в работе [1] предложена модель, названная CCS, которая строится на контейнере с возможностью произвольной вставки по индексу (например, контейнер list языка python). Все действия с контейнером описаны на псевдоязыке. Также в некоторых работах предлагаются сжатый диагональный (compressed diagonal, CDS) или skyline [2] способы представления.

Постановка задачи. В отличие от того, что предлагается в существующих публикациях, мы ставим своей целью построить архитектурный шаблон на основе имеющихся C++ классов библиотеки STL, позволяющий обеспечить оптимальное хранение разреженных матриц. Мы формулируем свою задачу именно как задачу сохранения синтаксически неизменного способа представления матрицы в виде двумерного массива, лишенного проблемы захвата избыточной оперативной памяти, работа с которым идет на основе оператора operator[]. Это позволит строить программный код в привычной для визуального представления манере и повысит его читаемость.

Разработка архитектурного шаблона. Для решения поставленной задачи мы предлагаем использовать статические переменные класса чисел с плавающей запятой, идентифицирующие последнее значение ключа и указатель на контейнер, к которому выполнялось последнее обращение с оператором operator[]. Новый тип данных с плавающей точкой должен уметь в рамках оператора присваивания различать ситуацию чтения из и записи в ассоциативный контейнер.

Диаграмма классов архитектурного шаблона приводится на рисунке 1.

Здесь важно пояснить, что нами для представления чисел с плавающей запятой введены новые классы fp и fp2. За счет переопределения операторов неявного преобразования типов числа классов fp и fp2 могут использоваться как обычные **double**-числа языка C++.

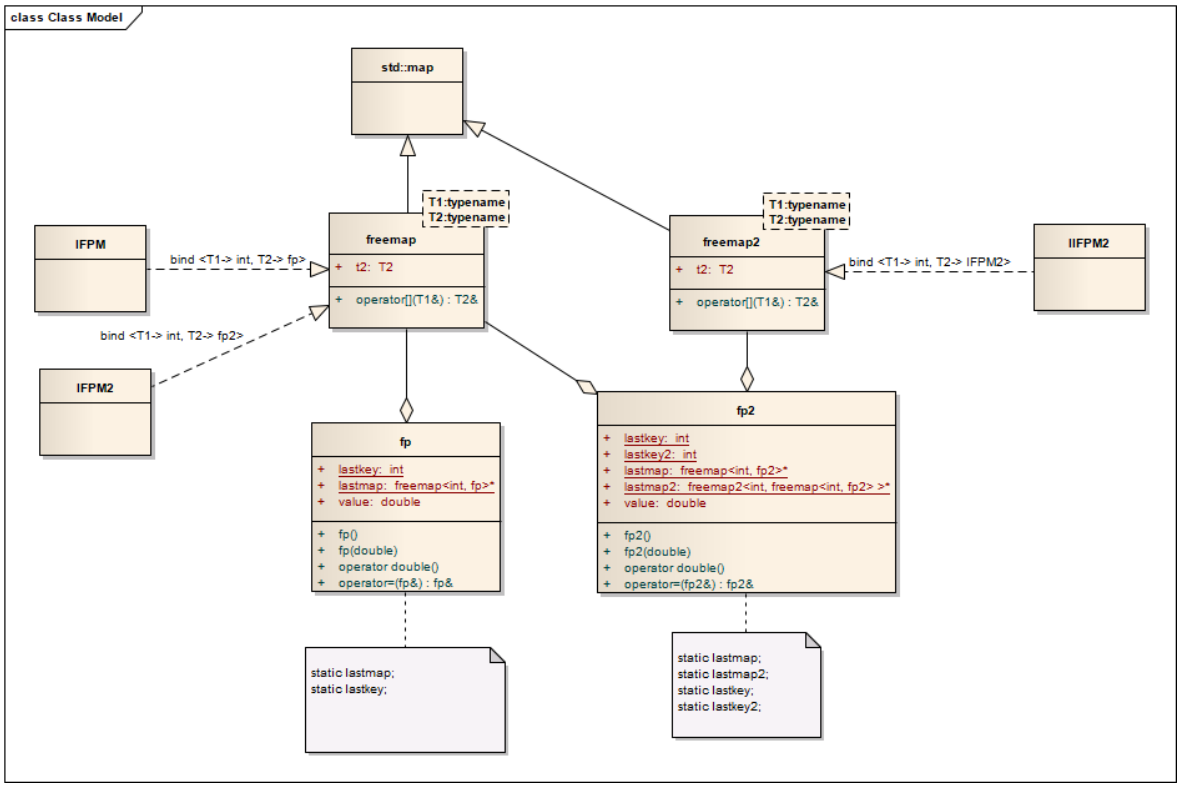

Рисунок 1. – Диаграмма классов архитектурного шаблона ассоциативного контейнера для разреженных матриц

В рамках предложенной кооперации классов создание ассоциативного контейнера будет выглядеть очень похоже на создание классического std::map-контейнера, но с обязательным определением статических переменных классов fp и fp2 в сегменте BSS:

```
typedef freemap<int, fp> IFPM;
IFPM *fp::lastmap = NULL;
int fp::lastkey;
typedef freemap<int, fp2> IFPM2;
IFPM2 *fp2::lastmap = NULL;
int fp2::lastkey;
typedef freemap2<int, IFPM2> IIFPM2;
IIFPM2 *fp2::lastmap2 = NULL;
int fp2::lastkey2;
```

Работа с такими типами ассоциативных контейнеров ничем не будет отличаться от контейнера std::map. Покажем это на примере:

```
IIFPM2 fpm;
// заносим 3 значения
  fpm[1][10] = 1.1;
  fpm[1][11] = 3.1;
  fpm[2][15] = 1.2;
  int s1 = fpm.size();          // вернет 2
  int s11 = fpm[1].size();    // вернет 2
  int s12 = fpm[2].size();    // вернет 1
  double n = fpm[1][11];    // в переменную n занесет 3.1
// считывание несуществующих элементов не добавляет их в контейнер
  double n1 = fpm[1][12]; // 0
  double n2 = fpm[3][13]; // 0
  double n3 = fpm[5][13]; // 0
  double n4 = fpm[6][13]; // 0
  double n5 = fpm[1][10]; // 1.1
  double n6 = fpm[2][15]; // 1.2
```

```
  fpm[2][15] = 120.134;  // меняем значение
  double n7 = fpm[2][15]; // вернет 120.134
// размеры векторов не изменились
  int s2 = fpm.size(); // 2
  int s21 = fpm[1].size(); // 2
  int s22 = fpm[2].size(); // 1
  int s32 = fpm[3].size(); // 0
  int s42 = fpm[4].size(); // 0
  int s52 = fpm[5].size(); // 0
  int s62 = fpm[6].size(); // 0
```

Обобщая полученные архитектурные решения для одномерного и двумерного ассоциативного контейнера на случай *n*-мерного контейнера, нам необходимо перейти к системе классов чисел с плавающей точкой fp, fp2,…, fpn и классов ассоциативных контейнеров IFPMn, IIFPMn,…, I…IFPMn.

В таком случае архитектурный шаблон может быть представлен диаграммой классов как на рисунке 2.

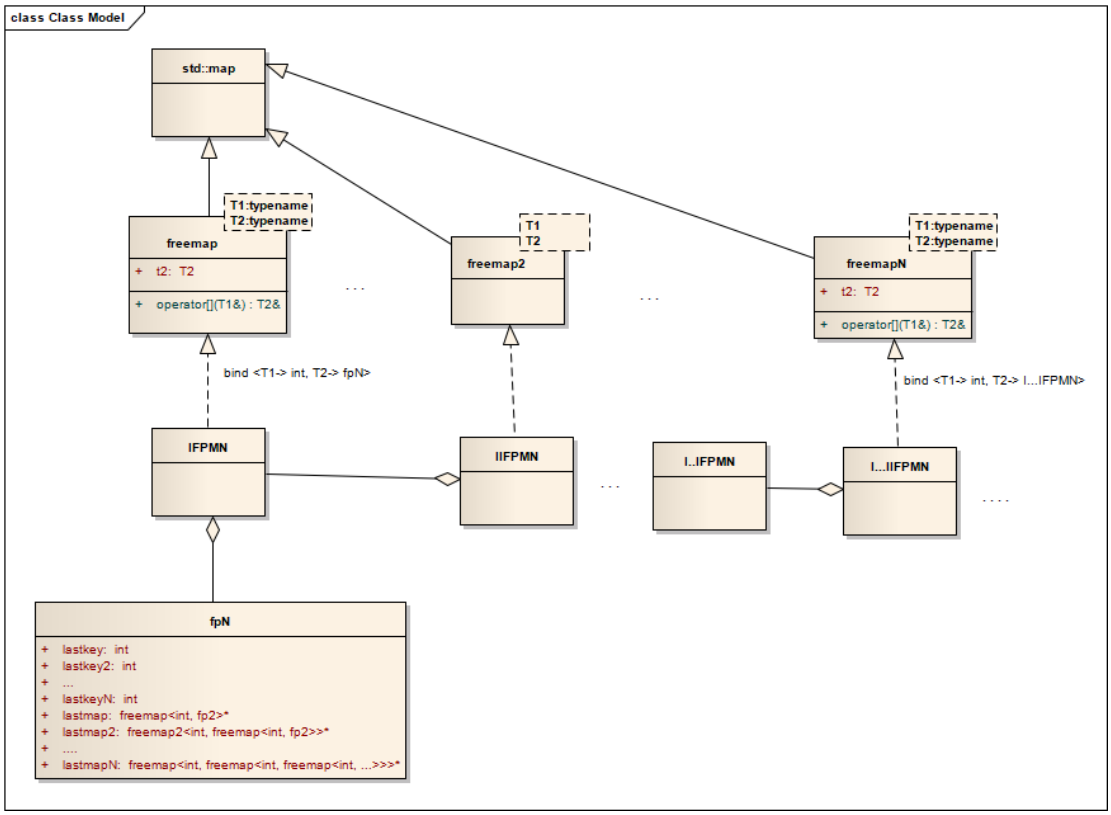

Рисунок 2. – Диаграмма классов архитектурного шаблона *n*-мерного ассоциативного контейнера для разреженных матриц

Реализация. Покажем ключевые аспекты реализации предложенной системы классов. Для классов, описывающих числа с плавающей точкой, важна инкапсуляция статических переменных для идентификации последнего вызова оператора operator[] у соответствующего ассоциативного контейнера, а также наличие перегруженных операторов неявного преобразования типов к типу **double**.

Приведем исходный код класса чисел с плавающей точкой для двумерного ассоциативного контейнера **fp2**:

```
class fp2 {
 public:
// статические переменные, идентифицирующие ассоциативные контейнеры
// первого и второго уровня, и соответствующие значения ключей i и j последнего обращения
// посредством оператора operator[] (например, m[i][j])
  static freemap<int, fp2>* lastmap;
  static freemap2<int, freemap<int, fp2> >* lastmap2;
  static int lastkey;
```

```
  static int lastkey2;
  double value;
// конструктор по умолчанию
  fp2() {
    value = 0;
    lastmap = NULL;
    lastmap2 = NULL;
  }
//  конструктор числа типа fp2 из типа double
fp2(double t) {
    value = t;
    lastmap = NULL;
    lastmap2 = NULL;
  }
// оператор неявного преобразования типа к типу double
  operator double() {
    return value;
  }
// оператор присваивания, различающий ситуацию записи/чтения ассоциативного контейнера
// по заполненности статических переменных
  fp2& operator=(const fp2 &p) {
    // если перед присваиванием срабатывал оператор operator[],
    // то создание объекта и помещение его в карту объектов первого уровня
    if (lastmap) {
      lastmap->insert(std::pair<int, fp2>(lastkey, p));
      //  создание карты второго уровня при необходимости
      if (lastmap2) {
        lastmap2->insert(std::pair<int, freemap<int, fp2> >(lastkey2, *lastmap));
        lastmap2->t2.clear();
      }
    }
    value = p;
    lastmap = NULL;
    lastmap2 = NULL;
    // возврат ссылки на объект после очистки статических переменных
    return *this;
  }
};
```

Для классов двумерных ассоциативных контейнеров необходимо реализовать два класса – **freemap** и **freemap2**. Приведем возможную реализацию данных классов:

```
template <typename T1, typename T2> class freemap: public std::map<T1, T2> {
public:
  T2 t2;
  T2& operator[] (const T1& k) {
     iterator it = find(k);
     if (it != end()) {
      return std::map<T1, T2>::operator[](k);
    }
    t2.lastmap = this;
    t2.lastkey = k;
    t2.value = 0;
    return t2;
  }
};
template <typename T1, typename T2> class freemap2: public std::map<T1, T2> {
public:
  T2 t2;
  T2& operator[] (const T1& k) {
```

```
    iterator it = find(k);
    if (it != end()) {
      return std::map<T1, T2>::operator[](k);
    }
    t2.t2.lastmap2 = this;
    t2.t2.lastkey2 = k;
    return t2;
  }
};
```

Как видно, одной из возможных реализаций оператора operator[] является реализация, основанная на вызове метода find() базового контейнера std::map.

Тестирование. Для предложенных реализаций архитектурного шаблона мы провели тестирование двумерного ассоциативного контейнера 10000×10000 элементов в сравнении с базовым контейнером класса std::map по времени заполнения на определенный процент и времени поэлементного считывания всего контейнера. Результаты приведены на рисунке 3.

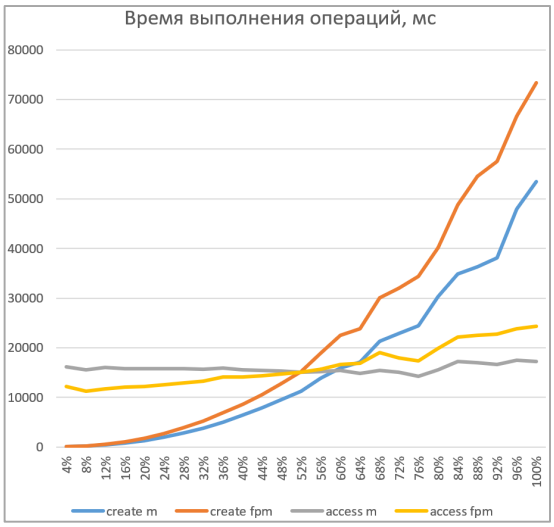

Рисунок 3. – Результаты времени заполнения и считывания контейнера m, основанного на классе std::map, и контейнера fpm, построенного на классах freemap и freemap2

На основании представленных графиков можно сделать вывод, что для разреженных матриц с заполнением менее 50% полученное архитектурное решение не только обеспечивает существенную экономию оперативной памяти, но и, вследствие более компактного хранения данных, дает выигрыш в быстродействии по доступу к элементам двумерного контейнера.

Матрицы жесткости метода конечных элементов строительных конструкций являются сильно разреженными (см. пример на рисунке 4).

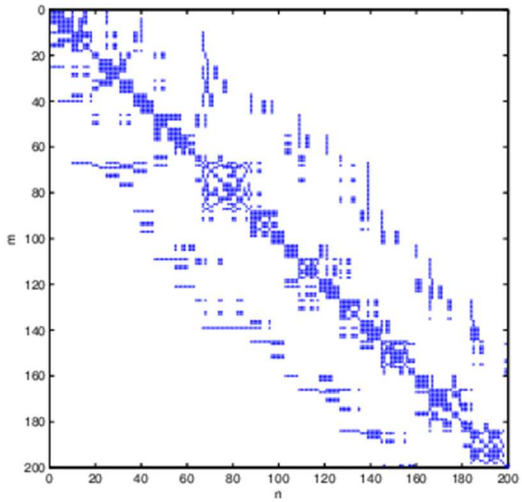

Рисунок 4. – Пример типовой матрицы жесткости метода конечных элементов [10] с заполнением 0,00327%

Предложенный архитектурный шаблон успешно решает поставленную задачу.

Заключение. В результате проведенного анализа были выявлены причины некорректной работы стандартных ассоциативных C++ контейнеров библиотеки STL. С целью устранения выявленных проблем разработан объектно-ориентированный архитектурный шаблон одно и двумерного ассоциативного контейнера для хранения разреженных матриц большой размерности.

Разработанный архитектурный шаблон применен нами в алгоритмах конечно-элементного анализа пространственных статически неопределимых строительных конструкций с учетом физической и геометрической нелинейности. На основе разработанного архитектурного шаблона построен класс двумерного ассоциативного контейнера для хранения разреженных матриц, содержащих числа с плавающей запятой удвоенной точности (тип double).

ЛИТЕРАТУРА

10. Богоявленский, А.И. Использование форматов хранения разреженных матриц при реализации метода конечных элементов / А.И. Богоявленский // Вестн. МГТУ им. Н.Э. Баумана. Сер. «Естественные науки». – 2017. – № 2. – С. 4–11.
11. Bathe, K.-J. Finite element procedures / K.-J. Bathe. – New Jersey : Prentice Hall, 1995. – 1037 p.
12. Кундас, С.П. Обзор численных методов расчета систем уравнений строительной механики и выбор оптимальной схемы хранения данных для задач большой размерности / С.П. Кундас, Д.О. Глухов, Т.М. Глухова // Вестн. Полоц. гос. ун-та. Сер. F, Строительство. Прикладные науки. – 2010. –№ 6. – С. 79–83.
13. Свириденко, А.Б. Прямые мультипликативные методы для разреженных матриц. Ньютоновские методы / Свириденко А.Б. // Компьютерные исследования и моделирование. – 2017. – Т. 9. – № 5. – С. 679–703.
14. Соловьев, С.А. Решение разреженных систем линейных уравнений методом Гаусса с использованием техники аппроксимации матрицами малого ранга / С.А. Соловьев // Вычислительные методы и программирование. – 2014. – Т. 15. – С. 441–460.
15. Отаров, А.О. Решение неустойчивых систем линейных алгебраических уравнений методом дифференциального спуска / А.О. Отаров, Э.П. Уразымбетова, А.А. Отаров // Вестн. Каракалпак. гос. ун-та им. Бердаха. – 2010. – № 3–4 (8–9). – С. 7–15.
16. Есаулов, В.А. Итерационный метод решения систем линейных уравнений с использованием q-градиента [Электронный ресурс] / Есаулов В.А., Д.В. Гринченков, В.А. Мохов // Инженер. вестн. Дона. – 2015. – № 3. – Режим доступа: https://cyberleninka.ru/article/n/iteratsionnyy-metod-resheniya-sistem-lineynyh-uravneniy-s-ispolzovaniem-q-gradienta.
17. Дмитриева, О.А. Оптимизация выполнения матрично-векторных операций при параллельном моделировании динамических процессов / О.А. Дмитриева // Наукові праці ДонНТУ. Сер. Обчислювальна техніка та автоматизація. – 2014. – № 1(26). – С. 94–100.
18. Писсанецки, С. Технология разреженных матриц / С. Писсанецки. – М. : Мир, 1988. – 410 с.
19. High-order unstructured methods for computational aero-acoustics / H. Beriot [et al.]. // Progress in simulation, control and reduction of ventilation noise / VKI. – Sint-Genesius-Rode, 2015.

Поступила 10.11.2020

SPARE MATRIX REPRESENTATION USING ASSOCIATIVE CONTAINERS C ++ STL LIBRARIES

D. GLUKHOV, R. BOGUSH, T. GLUKHOVA

The paper proposes an object-oriented architectural pattern for building associative containers designed to represent sparse matrices. The problem of classic associative STL containers is solved. This problem lies in the appearance of non-existent elements when you first access them. The proposed architectural pattern does not change the computational complexity of the basic operations of the std::map associative container. We have applied the proposed architectural template for a compact representation of a sparse stiffness matrix of a system of large-dimensional linear equations of the finite element method in the problem of calculating statically indeterminate building structures taking into account physical and geometric nonlinearity.

Keywords: *finite element method, sparse matrix, object-oriented design patterns, associative container.*