Содержание

Подп. и дата | Инв. № дубл. | Взам. инв. № | Подп. и дата | Инв. № подл.

Изм.	Лист	№ докум.	Подп.	Дата	Вариант №3		
Разраб.		Белым А.А.			Пояснительная записка к лабораторной работе по курсу «Вычислительный практикум» по теме «Суммирование разреженных матриц в формате RR(C)U»	Лит.	Лист 2 / Листов 27
Пров.		Ермаков А.С.					
Н. контр.						ТулГУ гр. 220601	
Утв.							

ВВЕДЕНИЕ

Очень часто программист работает с данными, заданными в виде матрицы (таблицы). Обработка таких данных требует довольно большого количества памяти, ведь приходится создавать двумерный массив большого размера. Однако не всегда вся исходная матрица заполнена нужными данными. Встречается такие ситуации, когда в огромной исходной таблице все элементы равны 0, за исключением малого количества. Такие матрицы называются разреженными. Хранить и обрабатывать такую таблицу в виде двумерного массива не оптимально. Желательно использовать что-то более рациональное.

В данной работе будет решаться проблема сложение разреженных матриц. Для этого будет использоваться специальный формат хранения разреженных матриц RR(C)U.

В данном отчёте сначала описывается сама задача, затем более подробно описывается про формат RR(C)U.

Далее описываются структуры данных и алгоритм для написания программы.

Отчёт также содержит полный текст программы на языках C и Python, описание всех функций, инструкцию пользователю и тестовый пример.

1. ЗАДАЧА

1.1. Содержательное описание задачи

Заданы две таблицы чисел, где большая часть элементов равна 0.

Требуется перевести две этих исходных матрицы в формат RR(C)U и сложить их в данном формате. Полученную матрицу вывести в виде таблицы, а также в формате RR(C)U.

Работа с разряженными матрицами может возникнуть в математическом анализе, а именно при решении дифференциальных уравнений в частных производных.

1.2. Формальная постановка задачи

Две разреженные матрицы, заданную в виде таблиц, требуется преобразовать в формат RR(C)U, сложить их и вывести на экран.

Осуществить вывод на экран как исходных, так и результирующих матриц в формате RR(C)U. Предусмотреть обнуление исходных матриц.

Рассмотрим формат представления разреженных матриц, т.е. матриц имеющих большое число нулевых элементов. В этом случае обычное представление матриц в виде массива будет избыточно, поэтому используются специальные форматы - RR(C)O и RR(C)U. Сокращенное название первого формата происходит от английского словосочетания "Row - wise Representation Complete and Ordered"(строчное представление, полное и упорядоченное). В данном формате вместо одного двумерного массива, используются три одномерных. Значения ненулевых элементов матрицы и соответствующие им столбовые индексы хранятся в этом формате по строкам в двух массивах AN и JA. Массив указателей IA, используется для ссылки на компоненты массивов AN и JA, с которых начинается описание очередной строки. Последняя компонента массива IA содержит указатель первой свободной компоненты в массивах AN и JA, т.е. равна числу ненулевых элементов матрицы, увеличенному

на единицу.

Сокращенное название второго формата происходит от английского словосочетания "Row - wise Representation Complete and Unordered"(строчное представление, полное, но неупорядоченное). Формат RR(C)U отличается от RR(C)O тем, что в данном случае соблюдается упорядоченность строк, но внутри каждой строки элементы исходных матриц могут храниться в произвольном порядке. Такие неупорядоченные представления могут быть весьма удобны в практических вычислениях. Результаты большинства матричных операций получаются неупорядоченными, а их упорядочение стоило бы значительных затрат машинного времени. В то же время, за немногими исключениями, алгоритмы для разреженных матриц не требуют, чтобы их представления были упорядоченными.

В общем случае описание r-й строки матрицы A хранится в компонентах с IA[r] до IA[r + 1]-1 массивов AN и JA. Если IA[r + 1] = IA[r], то это означает, что r - я строка нулевая. Количество элементов в массиве IA на единицу больше, чем число строк исходной матрицы, а количество элементов в массивах JA и AN равно числу ненулевых элементов исходной матрицы.

Рассмотренный формат называют полным, поскольку в нем указываются все ненулевые элементы матрицы A, упорядоченным, поскольку элементы каждой строки матрицы A хранятся по возрастанию столбовых индексов, и строчным, поскольку информация о матрице A указывается по строкам.

Говорят, что массивы IA и JA представляют портрет (структуру) матрицы A. Если алгоритм, реализующий какую - либо операцию над разреженными матрицами, разбит на этапы символической обработки, на котором определяется портрет результирующей матрицы, и численной обработки, на котором определяются значения элементов результирующей матрицы, то массивы IA и JA заполняются на первом этапе, а массив AN - на втором.

Рассмотрим теперь алгоритмы сложения матриц в формате RR(C)U.

На вход подаются переменные N,M - соответственно количество строк и столбцов матриц, IA,JA,AN - массивы используемые в представлении RR(C)U матрицы A.

IB,JB,AB - массивы используемые в представлении RR(C)U матрицы B. На выходе получаем массивы IC,JC,CN содержащий искомую матрицу в форме RR(C)U.

Алгоритм вначале формирует портрет матрицы C в массивах IC,JC, а затем заполняет массив CN значениями ненулевых элементов матрицы C. Можно исправить алгоритм сделав так, чтобы формирование портрета матрицы и заполнение массива CN проводилось одновременно, именно так устроен алгоритм, предъявленный ниже.

Есть одна маленькая проблема в работе алгоритма, а именно, если для некоторых i,j выполняется a[i,j]=-b[i,j] < >0, то в представлении результирующей матрицы элемент c[i,j] должен отсутствовать, но данный алгоритм не отслеживает эту ситуацию, поэтому возможно возникновение нулевых элементов в массиве CN.

Эта проблема решается во втором варианте алгоритма сложения RR(C)U-матриц.

В отличии от предыдущего, данный алгоритм заполняет массивы IC,JC,NC за один проход, к тому же он проверяет возникновение ситуации, когда a[i,j]=-b[i,j] < >0 и не допускает появления в массиве CN нулевых элементов, правда это скажется на скорости работы.

Для этого алгоритм сначала проходит по строке матрицы A. Если в соответствующей строке матрицы B в массиве JB есть такой же элемент, как и в JA, то строки и столбцы этих элементов совпадают, их сумма проверяется на равенство 0, и в случае неравенства добавляется в матрицу C. Если в JB не найдено соответствующего индекса строки, то элемент из A просто добавляется в матрицу C. Добавленные индексы строк помечаются как "использованные". После того, как пройдена вся строка в матрице A, алгоритм проходит строку в матрице B; если текущий элемент JB не отмечен как "использованный то он добавляется в матрицу C. После прохода по строке матрицы B алгоритм очищает список использованных вершин и переходит к следующей строке.

2. РАЗРАБОТКА АЛГОРИТМА

2.1. Разработка графического интерфейса пользователя

Для ввода исходных матриц требуются две таблицы. Количество строк и столбцов должен вводить пользователь. Изменение количества строк или столбцов непосредственно в самой таблице должны вступать в силу после того, как пользователь нажмет кнопку "Новая матрица тогда создается новая нулевая матрица указанного размера, или кнопку "Изменить размер в данном случае уже введенная информация сохраняется.

Для вывода результирующей матрицы предусмотреть ещё одну таблицу. Запретить её редактирование.

Под каждой матрицей необходимо добавить три текстовых поля, для вывода матриц в формате RR(C)U (т.е. отдельные таблицы под массивы IA, JA, AN).

Создать панель меню со следующими разделами:

1) Файл. Содержит разделы: "Выход";

2) Правка. Содержит раздел "Создать пустые матрицы "Изменить размер матриц".

3) Запуск. Содержит раздел "Создать RR(C)U матрицы A "Создать RR(C)U матрицы B "Сложить RR(C)U матриц A и B".

4) Справка. Содержит раздел "Справка".

Итак, внешний вид разработанного интерфейса представлен на рисунке 1.

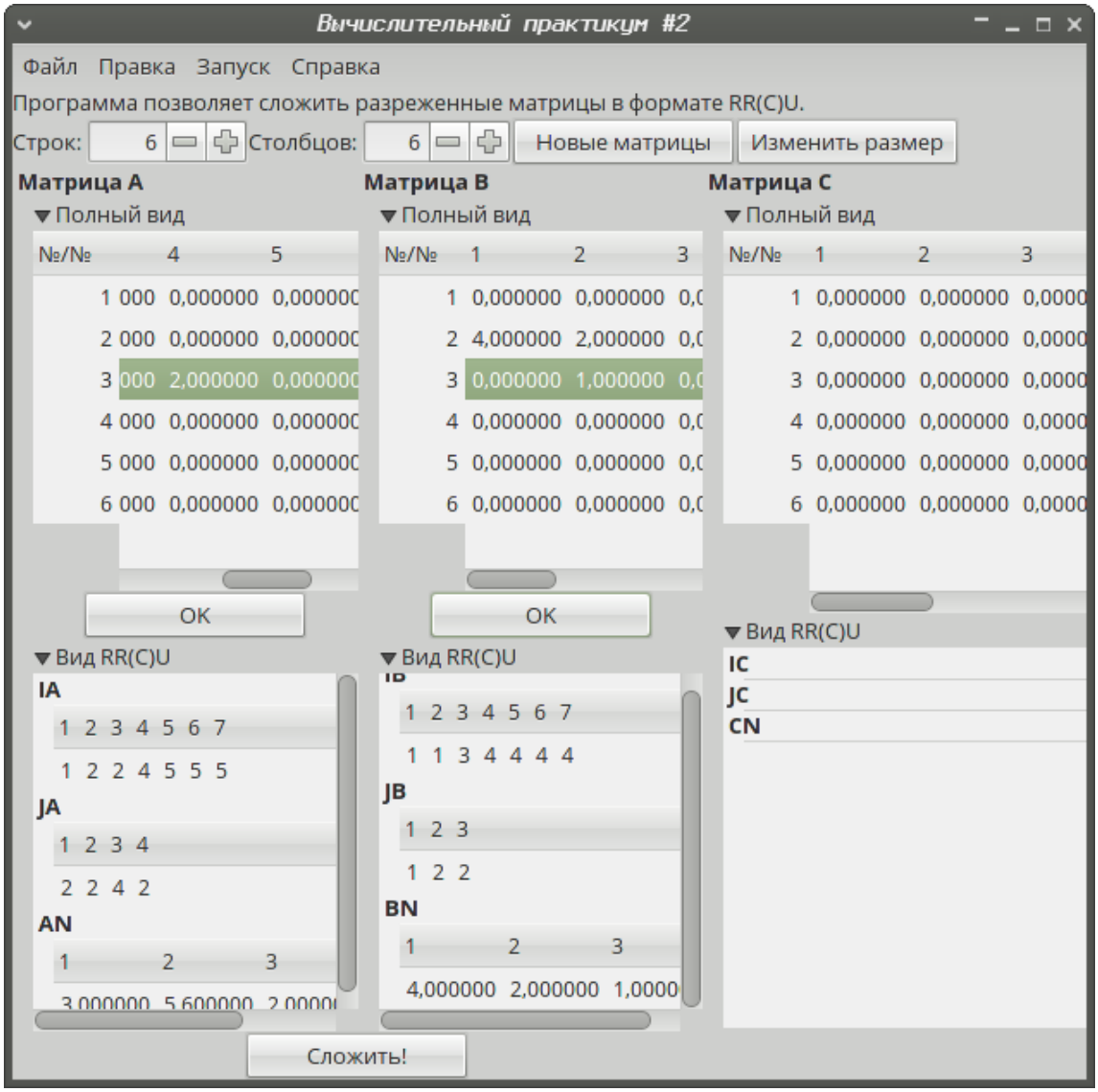

Рисунок 1 – Разработанный интерфейс программы

2.2. Разработка структур данных

Разреженная матрица в формате RR(C)U хранится в виде трёх массивов, следовательно, разумно создать структуру данных (RRCU) со следующими полями:

rows_ptrs (IA) – массив типа int;

cols_nums (JA) – массив типа int;

elements (AN) – массив типа float;

Этот тип описывает переменные, в которых хранятся две исходные матрицы, и матрица их суммы.

2.3. Разработка структуры алгоритма

Основную программу можно разбить на три участка: считывание значений с таблиц и сохранение их в матрицы формата RR(C)U, суммирование матриц и вывод полученной матрицы из формата RR(C)U в таблицу.

Для суммирования матриц будет создана подпрограмма sum_rrcu, принимающая 2 параметра: матрицы типа RRCU, которые надо сложить.

2.4. Схема алгоритма

На рисунке 2 представлена схема алгоритма суммирования двух матриц в формате RR(C)U.

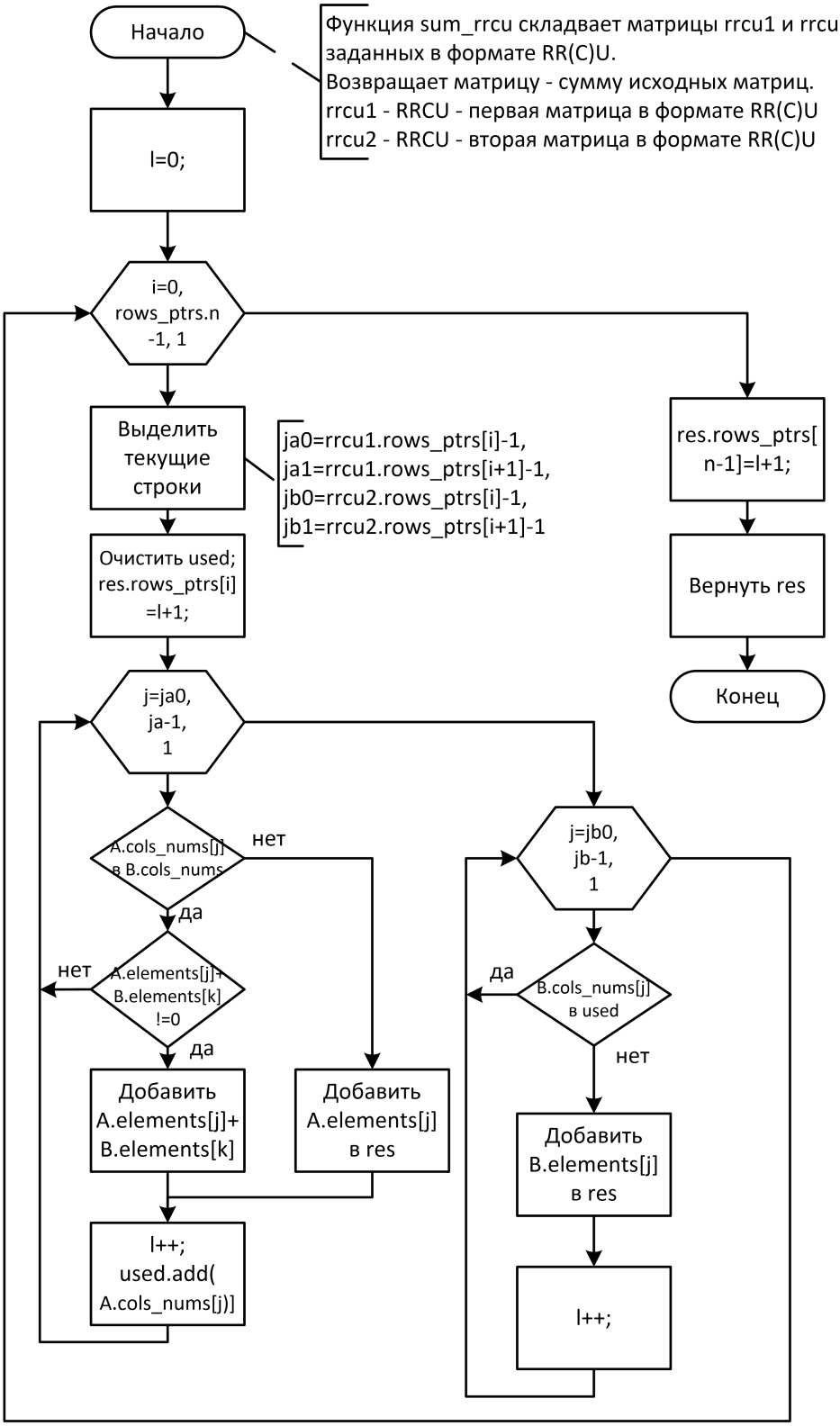

Рисунок 2 – Схема алгоритма суммирования матриц формата RR(C)U

3. РАЗРАБОТКА ПРОГРАММЫ

3.1. Описание переменных и структур данных

В данной программе используются следующая структура:

RRCU – структура для хранения матрицы в формате RR(C)U со следующими полями:

rows_ptrs (IA) – массив типа VECTORi;

cols_nums (JA) – массив типа VECTORi;

elements (AN) – массив типа VECTORf;

Все массивы определяются типом VECTORx, который содержит в себе:

n - int - количество элементов,

elements - массив из float(x==f) или int(x==i) - элементы массива.

Матрицы стандартного вида задаются типом MATRIX:

n,m - int - количество строк и столбцов,

elements - массив из float - элементы матрицы.

3.2. Описание функций

1. RRCU sum_rrcu(RRCU rrcu1, RRCU rrcu2)

Функция sum_rrcu складывает матрицы rrcu1 и rrcu2, заданных в формате RR(C)U.

Возвращает матрицу - сумму исходных матриц.

Параметры функции представлены в таблице 1 :

Таблица 1 – Параметры функции сложения RR(C)U

имя	тип	предназначение
rrcu1	RRCU	первая матрица в формате RR(C)U
rrcu2	RRCU	вторая матрица в формате RR(C)U

2. MATRIX expand_rrcu(RRCU rrcu,int m)

Функция expand_rrcu восстанавливает стандартное представление матрицы из матрицы rrcu, заданной в формате RR(C)U, с m столбцами.

Возвращает матрицу в стандартном представлении. Параметры функции представлены в таблице 2 :

Таблица 2 – Параметры функции восстановления из RR(C)U

имя	**тип**	**предназначение**
rrcu	RRCU	матрица в формате RR(C)U
m	int	количество столбцов в полном представлении матрицы.

3. RRCU create_rrcu(MATRIX a)

Функция create_rrcu преобразует матрицу a в формат разреженных матриц RR(C)U.

Возврщает матрицу в формате RR(C)U. Параметры функции представлены в таблице 3 :

Таблица 3 – Параметры функции получения RR(C)U

имя	**тип**	**предназначение**
a	MATRIX	матрица в стандартном представлении.

4. input_data(self):

Вводит исходные данные.

5. output_data(self,rrcu_c)

Выводит полученные данные.

6. on_run_click(self,button,data=None):

Производит считывание данных, отсечение и вывод результатов.

4. ИНСТРУКЦИЯ ПОЛЬЗОВАТЕЛЮ

Данная программа транспонирует заданную матрицу и выводит её на экран.

Данная программа не требует установки. Для её запуска необходимо открыть файл prac2.py. Внимание: для работы приложения на компьютере должен быть установлен Python 3, GTK+3, GObject-introspection.

Для начала требуется задать кол-во строк и столбцов в исходных матрицах. Эти числа не могут быть меньше 2. После этого нужно нажать кнопку "Создать матрицы что приведет к созданию пустых матриц указанного размера, либо "Изменить размер что изменит размер матриц и при этом сохранит уже введенные данные, если они входят в новые размеры.

После ввода значений в таблицу следует выбрать пункт меню "Создать RR(C)U матрицы A "Создать RR(C)U матрицы B или нажать кнопку под соотв. таблицей. После этого можно суммировать матрицы с помощтю меню "Сложить RR(C)U матриц A и B"или же нажать соотв. кнопку. После этого на экран будет выведена матрица суммы исходных матриц.

Внизу таблиц выводятся массивы, описывающие эти матрицы в формате RR(C)U.

5. ТЕСТОВАЯ ЗАДАЧА

5.1. Аналитическое решение и умозрительные результаты

Возьмём матрицы 10x10. Матрица A:

$$\begin{matrix}
1 & 0 & 0 & 0 & 0 & 3 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{matrix}$$

Матрица B:

$$\begin{matrix}
1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 6 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{matrix}$$

После сложения результат будет иметь вид:

$$\begin{matrix}
2 & 0 & 0 & 0 & 0 & 3 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 6 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 12 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{matrix}$$

5.2. Решение, полученное с использованием разработанного ПО

Ниже на рисунке 3 представлен пример работы программы сложения матриц в формате RR(C)U.

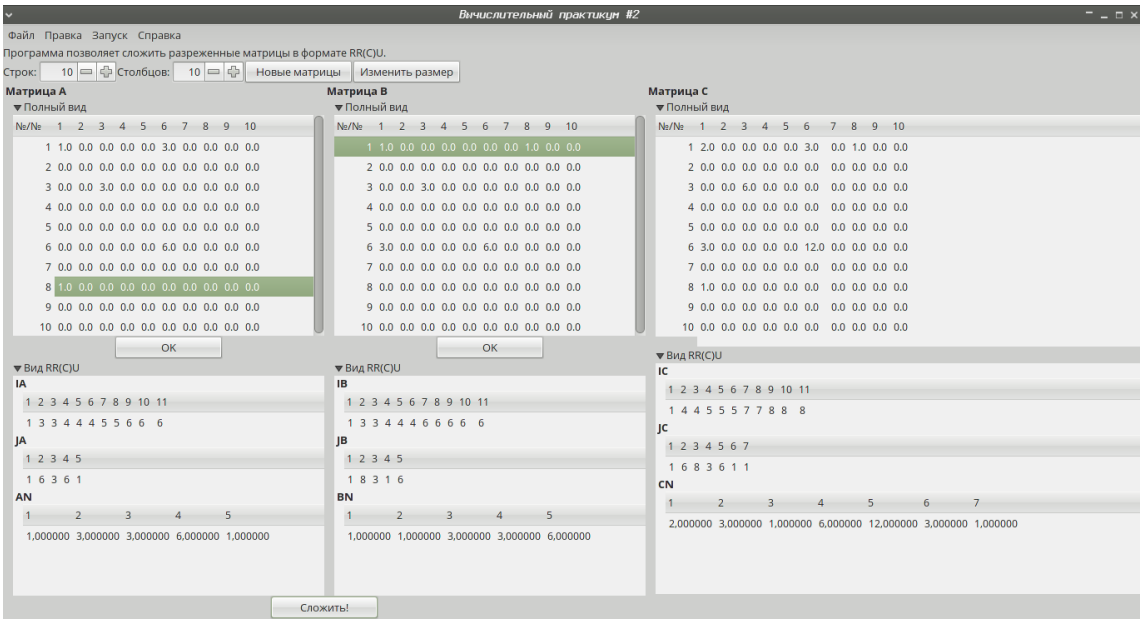

Рисунок 3 – Пример работы программы сложения RR(C)U-матриц

5.3. Выводы

Данная программа производит сложение матриц. Введённые значения в таблицы программа преобразует в формат RR(C)U, затем складывает матрицы в этом формате, а затем полученную матрицу из формата RR(C)U выводит в таблицу.

ЗАКЛЮЧЕНИЕ

Разреженные матрицы требуют особого способа хранения, так как хранение их в виде массива нерационально. В написанной программе разрежённые матрицы хранятся в формате RR(C)U, в нём же и выполняются все вычисления (сложение матриц). Данный формат предоставляет экономию ресурсов и времени при работе с такими матрицами.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. http://python.org
2. http://www.gtk.org
3. http://ru.wikipedia.org
4. http://en,wikipedia.org

ПРИЛОЖЕНИЕ

Ниже приведен текст модуля расширения Python, реализующего работу с разреженными матрицами в формате RR(C)U и написанного на Си.

```c
#include <Python.h>
#include <structmember.h>

typedef struct{
    double *elements;
    int n,m;
} MATRIX;

typedef struct{
    double *elements;
    int n;
} VECTOR_f;

typedef struct{
    long *elements;
    int n;
} VECTOR_l;

typedef struct{
    VECTOR_l cols_nums,rows_ptrs;
    VECTOR_f elements;
} RRCU;

int x_in_v(double x,VECTOR_l v,long i0,long i1){
    printf("i1=%ld\n",i1);
    while (i1>=i0&&v.elements[i1]!=x)
        --i1;
    return i1;
}
void print_vf(VECTOR_f v){
    int i;
    for (i=0;i<v.n;++i){
        printf("%f ",v.elements[i]);
    }
    printf("\n");
}


void print_vl(VECTOR_l v){
    int i;
    for (i=0;i<v.n;++i){
        printf("%ld ",v.elements[i]);
    }
    printf("\n");
}
/*
Функция create_rrcu преобразует матрицу a в формат разреженных матриц RR(C)U.
Возвращает матрицу в формате RR(C)U.
Параметры:
a — MATRIX — матрица в стандартном представлении.
*/
RRCU create_rrcu(MATRIX a){
    int i,j,l=0;
    RRCU res;
    res.elements.n=0;res.elements.elements=NULL;
    res.rows_ptrs.n=a.n+1;res.rows_ptrs.elements=malloc((a.n+1)*sizeof(long));
    res.cols_nums.n=0; res.cols_nums.elements=NULL;
    for (i=0;i<a.n;++i){
        res.rows_ptrs.elements[i]=l+1;
```

```
        for (j=0;j<a.m;++j){
            if (a.elements[i*a.m+j]!=0.0){
                ++res.elements.n;
                res.elements.elements=realloc(res.elements.elements,
                        res.elements.n*sizeof(double));
                res.elements.elements[l]=a.elements[i*a.m+j];
                ++res.cols_nums.n;
                res.cols_nums.elements=realloc(res.cols_nums.elements,
                        res.cols_nums.n*sizeof(long));
                res.cols_nums.elements[l]=j+1;
                ++l;
            }
        }
    }
    res.rows_ptrs.elements[a.n]=l+1;
    return res;
}
/*
Функция expand_rrcu восстанавливает стандартное представление матрицы из
матрицы rrcu, заданной в формате RR(C)U, с m столбцами.
Возвращает матрицу в стандартном представлении.
Параметры:
rrcu — RRCU — матрица в формате RR(C)U
m — количество столбцов в полном представлении матрицы.
*/
MATRIX expand_rrcu(RRCU rrcu,int m){
    MATRIX res;
    long i,j;
    res.n=rrcu.rows_ptrs.n-1;
    res.m=m;
    res.elements=calloc(res.n*res.m,sizeof(double));
    for (i=0;i<res.n;++i){
        for (j=rrcu.rows_ptrs.elements[i]-1;
             j<rrcu.rows_ptrs.elements[i+1]-1;
             ++j){
            res.elements[i*res.m+rrcu.cols_nums.elements[j]-1]=
                                            rrcu.elements.elements[j];
        }
    }
    return res;
}
/*
Функция sum_rrcu складвает матрицы rrcu1 и rrcu2, заданных в формате RR(C)U.
Возвращает матрицу — сумму исходных матриц.
rrcu1 — RRCU — первая матрица в формате RR(C)U
rrcu2 — RRCU — вторая матрица в формате RR(C)U
*/
RRCU sum_rrcu(RRCU rrcu1, RRCU rrcu2){
    RRCU res;
    res.rows_ptrs.n=rrcu1.rows_ptrs.n;
    res.rows_ptrs.elements=malloc((rrcu1.rows_ptrs.n)*sizeof(long));
    res.cols_nums.n=0;
    res.cols_nums.elements=NULL;
    res.elements.n=0;
    res.elements.elements=NULL;
    print_vl(rrcu1.rows_ptrs);
    print_vl(rrcu2.rows_ptrs);
    long l=0;
    int i,j,k;
```

```
VECTOR_l used;
for (i=0;i<rrcu1.rows_ptrs.n-1;i++){
    res.rows_ptrs.elements[i]=l+1;
    printf("1\n----------\n");
    int ja0=rrcu1.rows_ptrs.elements[i]-1,
        ja1=rrcu1.rows_ptrs.elements[i+1]-1,
        jb0=rrcu2.rows_ptrs.elements[i]-1,
        jb1=rrcu2.rows_ptrs.elements[i+1]-1;
    used.n=0;
    printf("ja0=%d ja1=%d\n",ja0,ja1);
    used.elements=NULL;
    for (j=ja0;j<ja1;++j){
        printf("2\n");
        if ((k=x_in_v(rrcu1.cols_nums.elements[j],
                      rrcu2.cols_nums,
                      jb0,jb1-1))>=jb0){

            if (rrcu2.elements.elements[k]!=-rrcu1.elements.elements[j]){
                printf("31\n");
                ++res.elements.n;
                res.elements.elements=realloc(res.elements.elements,
                                    res.elements.n*sizeof(double));
                res.elements.elements[l]=rrcu1.elements.elements[j]+
                                        rrcu2.elements.elements[k];
                ++res.cols_nums.n;
                res.cols_nums.elements=realloc(res.cols_nums.elements,
                                        res.cols_nums.n*sizeof(long));
                res.cols_nums.elements[l]=rrcu1.cols_nums.elements[j];
                ++l;
            }
            ++used.n;
            used.elements=realloc(used.elements,used.n*sizeof(long));
            used.elements[used.n-1]=rrcu1.cols_nums.elements[j];
        }else{
            printf("4\n");
            ++res.elements.n;
            res.elements.elements=realloc(res.elements.elements,
                                    res.elements.n*sizeof(double));
            res.elements.elements[l]=rrcu1.elements.elements[j];
            printf("41\n");
            ++res.cols_nums.n;
            res.cols_nums.elements=realloc(res.cols_nums.elements,
                                    res.cols_nums.n*sizeof(long));
            res.cols_nums.elements[l]=rrcu1.cols_nums.elements[j];
            ++l;
        }
    }
    for (j=jb0;j<jb1;++j){
        printf("5\njb0=%d jb1=%d used.n=%d\n",jb0,jb1,used.n);
        if (x_in_v(rrcu2.cols_nums.elements[j],
                   used,
                   0,used.n-1)<0){
            printf("6\n");
            ++res.elements.n;
            res.elements.elements=realloc(res.elements.elements,
                                res.elements.n*sizeof(double));
            res.elements.elements[l]=rrcu2.elements.elements[j];
            ++res.cols_nums.n;
            res.cols_nums.elements=realloc(res.cols_nums.elements,
```

```
                                                res.cols_nums.n*sizeof(long));
                    res.cols_nums.elements[l]=rrcu2.cols_nums.elements[j];
                    ++l;
                }
                printf("50\n");
            }
            free(used.elements);
        }
        printf("7\n");
        print_vl(res.rows_ptrs);
        res.rows_ptrs.elements[res.rows_ptrs.n-1]=l+1;
        printf("8\n");
        return res;
    }
    VECTOR_f PyList_to_VECTORf(PyObject* list){
        VECTOR_f res;
        int i;
        res.n=PyList_Size(list);
        res.elements=malloc(res.n*sizeof(double));
        for (i=0;i<res.n;++i){
            res.elements[i]=PyFloat_AsDouble(PyList_GetItem(list,i));
        }
        return res;
    }
    PyObject* VECTORf_to_PyList(VECTOR_f v){
        PyObject* list=PyList_New(v.n);
        int i;
        for (i=0;i<v.n;++i){
            PyList_SetItem(list,i,PyFloat_FromDouble(v.elements[i]));
        }
        return list;
    }

    VECTOR_l PyList_to_VECTORl(PyObject* list){
        VECTOR_l res;
        int i;
        res.n=PyList_Size(list);
        res.elements=malloc(res.n*sizeof(long));
        for (i=0;i<res.n;++i){
            res.elements[i]=PyLong_AsLong(PyList_GetItem(list,i));
        }
        return res;
    }
    PyObject* VECTORl_to_PyList(VECTOR_l v){
        PyObject* list=PyList_New(v.n);
        int i;
        for (i=0;i<v.n;++i){
            PyList_SetItem(list,i,PyLong_FromLong(v.elements[i]));
        }
        return list;
    }

    MATRIX PyList_to_MATRIX(PyObject* list){
        int i,j;
        MATRIX res;
        res.n=PyList_Size(list);
        PyObject *line=PyList_GetItem(list,0);
        res.m=PyList_Size(line);
        res.elements=malloc(res.m*res.n*sizeof(double));
```

```
    for (i=0;i<res.n;i++){
        line=PyList_GetItem(list,i);
        for (j=0;j<res.m;j++){
            res.elements[i*res.m+j]=PyFloat_AsDouble(PyList_GetItem(line,j));
        }
    }
    return res;
}

PyObject* MATRIX_to_PyList(MATRIX matrix){
    PyObject* res=PyList_New(matrix.n);
    int i,j;
    for (i=0;i<matrix.n;++i){
        PyObject* line=PyList_New(matrix.m);
        PyList_SetItem(res,i,line);
        for(j=0;j<matrix.m;++j){
            PyList_SetItem(line,j,PyFloat_FromDouble(matrix.elements[i*matrix.m+j]));
        }
    }
    return res;
}

typedef struct {
    PyObject_HEAD;
    /* Type-specific fields go here. */
    PyObject *rows_ptrs,*cols_nums, *elements;
} Py_RRCU;

static void
Py_RRCU_dealloc(Py_RRCU* self)
{
    Py_XDECREF(self->rows_ptrs);
    Py_XDECREF(self->cols_nums);
    Py_XDECREF(self->elements);
    Py_TYPE(self)->tp_free((PyObject*)self);
}

static PyObject *
Py_RRCU_new(PyTypeObject *type, PyObject *args, PyObject *kwds)
{
    Py_RRCU *self;
    printf("New X\n");
    self = (Py_RRCU *)type->tp_alloc(type, 0);
    if (self != NULL) {
        self->rows_ptrs = Py_None;
        if (self->rows_ptrs == NULL)
          {
            Py_DECREF(self);
            return NULL;
          }
        Py_INCREF(Py_None);
        self->cols_nums = Py_None;
        if (self->cols_nums == NULL)
          {
            Py_DECREF(self);
            return NULL;
          }
        Py_INCREF(Py_None);
        self->elements = Py_None;
```

```
        if (self->cols_nums == NULL)
          {
            Py_DECREF(self);
            return NULL;
          }
    }

    return (PyObject *)self;
}

static int
Py_RRCU_init(Py_RRCU *self, PyObject *args, PyObject *kwds)
{
   // PyObject *rows_ptrs,*cols_nums,*elements;
    PyObject *py_matrix=Py_None;
    MATRIX matrix;
    RRCU new_rrcu;
    printf("init 0\n");
    if (! PyArg_ParseTuple(args,"|O",&py_matrix))
        return -1;
    Py_DECREF(self->elements);
    Py_DECREF(self->cols_nums);
    Py_DECREF(self->rows_ptrs);
    if (PyObject_IsTrue(py_matrix)&&PyList_Check(py_matrix)){
        matrix=PyList_to_MATRIX(py_matrix);
        new_rrcu=create_rrcu(matrix);
        free(matrix.elements);
        self->elements=VECTORf_to_PyList(new_rrcu.elements);
        self->rows_ptrs=VECTORl_to_PyList(new_rrcu.rows_ptrs);
        self->cols_nums=VECTORl_to_PyList(new_rrcu.cols_nums);
        free(new_rrcu.elements.elements);
        free(new_rrcu.rows_ptrs.elements);
        free(new_rrcu.cols_nums.elements);
    }else{
        self->elements=PyList_New(0);
        self->cols_nums=PyList_New(0);
        self->rows_ptrs=PyList_New(0);
    }
    printf("Init X\n");
    return 0;
}


static PyMemberDef Py_RRCU_members[] = {
    {"rows_ptrs", T_OBJECT_EX, offsetof(Py_RRCU,rows_ptrs ), 0,
     "first name"},
    {"cols_nums", T_OBJECT_EX, offsetof(Py_RRCU,cols_nums ), 0,
     "last name"},
    {"elements", T_OBJECT_EX, offsetof(Py_RRCU,elements ), 0,
     "noddy number"},
    {NULL}  /* Sentinel */
};

static PyTypeObject Py_RRCUType = {
    PyVarObject_HEAD_INIT(NULL, 0)
    "alg2.RRCU",             /* tp_name */
    sizeof(Py_RRCU), /* tp_basicsize */
    0,                         /* tp_itemsize */
    (destructor)Py_RRCU_dealloc,  /* tp_dealloc */
```

```
    0,                          /* tp_print */
    0,                          /* tp_getattr */
    0,                          /* tp_setattr */
    0,                          /* tp_reserved */
    0,                          /* tp_repr */
    0,                          /* tp_as_number */
    0,                          /* tp_as_sequence */
    0,                          /* tp_as_mapping */
    0,                          /* tp_hash  */
    0,                          /* tp_call */
    0,                          /* tp_str */
    0,                          /* tp_getattro */
    0,                          /* tp_setattro */
    0,                          /* tp_as_buffer */
    Py_TPFLAGS_DEFAULT,         /* tp_flags */
    "RRCU sparse matrix",           /* tp_doc */
    0,                      /* tp_traverse */
    0,                      /* tp_clear */
    0,                      /* tp_richcompare */
    0,                      /* tp_weaklistoffset */
    0,                      /* tp_iter */
    0,                      /* tp_iternext */
    0,                      /* tp_methods */
    Py_RRCU_members,            /* tp_members */
    0,                          /* tp_getset */
    0,                          /* tp_base */
    0,                          /* tp_dict */
    0,                          /* tp_descr_get */
    0,                          /* tp_descr_set */
    0,                          /* tp_dictoffset */
    (initproc)Py_RRCU_init,     /* tp_init */
    0,                          /* tp_alloc */
    Py_RRCU_new,
};

static PyObject* expand_rrcu_wrapper(PyObject *self, PyObject *args){
    long m;
    MATRIX matrix;
    PyObject *res;
    Py_RRCU *py_rrcu;
    RRCU rrcu;
    PyArg_ParseTuple(args,"Oi",&py_rrcu,&m);

    rrcu.cols_nums=PyList_to_VECTORl(py_rrcu->cols_nums);
    rrcu.rows_ptrs=PyList_to_VECTORl(py_rrcu->rows_ptrs);
    rrcu.elements=PyList_to_VECTORf(py_rrcu->elements);

    matrix=expand_rrcu(rrcu,m);
    res=MATRIX_to_PyList(matrix);

    free(matrix.elements);
    free(rrcu.elements.elements);
    free(rrcu.rows_ptrs.elements);
    free(rrcu.cols_nums.elements);

    return res;
}

PyObject* sum_rrcu_wrapper(PyObject* self,PyObject *args){
```

```
    RRCU a,b,c;
    Py_RRCU *py_a,*py_b,*py_c;
    PyArg_ParseTuple(args,"OO",&py_a,&py_b);
    a.cols_nums=PyList_to_VECTORl(py_a->cols_nums);
    a.rows_ptrs=PyList_to_VECTORl(py_a->rows_ptrs);
    a.elements=PyList_to_VECTORf(py_a->elements);
    b.cols_nums=PyList_to_VECTORl(py_b->cols_nums);
    b.rows_ptrs=PyList_to_VECTORl(py_b->rows_ptrs);
    b.elements=PyList_to_VECTORf(py_b->elements);
    c=sum_rrcu(a,b);

    PyObject* temp_args=Py_BuildValue("(O)",Py_None);
    py_c=(Py_RRCU*)PyObject_CallObject((PyObject*)&Py_RRCUType,temp_args);
    Py_XDECREF(temp_args);
    printf("A\n");
    py_c->rows_ptrs=VECTORl_to_PyList(c.rows_ptrs);
    py_c->cols_nums=VECTORl_to_PyList(c.cols_nums);
    py_c->elements=VECTORf_to_PyList(c.elements);
    printf("B\n");
    free(c.rows_ptrs.elements);free(c.cols_nums.elements);free(c.elements.elements);
    free(b.rows_ptrs.elements);free(b.cols_nums.elements);free(b.elements.elements);
    free(a.rows_ptrs.elements);free(a.cols_nums.elements);free(a.elements.elements);

    return (PyObject*)py_c;
}

static PyMethodDef alg2methods[]={

    {"expand_rrcu",expand_rrcu_wrapper,METH_VARARGS,""},
    {"sum_rrcu",sum_rrcu_wrapper,METH_VARARGS,""},
    {NULL,NULL,0,NULL}
};
static PyModuleDef alg2module = {
    PyModuleDef_HEAD_INIT,
    "alg2",
    "RRCU sparse matrix algorhytms",
    -1,
    alg2methods
};

PyMODINIT_FUNC
PyInit_alg2(void)
{
    PyObject* m;

    if (PyType_Ready(&Py_RRCUType) < 0)
        return NULL;

    m = PyModule_Create(&alg2module);
    if (m == NULL)
        return NULL;

    Py_INCREF(&Py_RRCUType);
    PyModule_AddObject(m, "RRCU", (PyObject*)&Py_RRCUType);
    return m;
}
```

Далее приводится текст основной программы, написанной на Python 3.

```
#!/usr/bin/env python
from gi.repository import Gtk
from locale import atof,atoi
from alg2 import RRCU,expand_rrcu,sum_rrcu

class Application(Gtk.Builder):
    def __init__(self,ui_filename):
        Gtk.Builder.__init__(self)
        self.add_from_file(ui_filename)
        self.connect_signals(self)
        self.liststoresA=[]
        self.liststoresB=[]
        self.liststoresC=[]
        self.update_tables()
    def show(self,form_name):
        window = self.get_object(form_name)
        window.show()
        Gtk.main()
    def show_msg(self,msg):
        md=Gtk.MessageDialog(None, Gtk.DialogFlags.MODAL,
                             Gtk.MessageType.WARNING, Gtk.ButtonsType.OK, msg);
        md.run ();
        md.destroy();
    def clear_table(self,table):
        model=table.get_model()
        if model:
            model.clear()
        try:
            c=table.get_column(0)
            while c:
                table.remove_column(c)
                c=table.get_column(0)
        except:
            pass
    def on_resize_click(self,widget,data=None):
        for table in (self.get_object("treeview1"),
                      self.get_object("treeview2"),
                      self.get_object("treeview3"),
                      self.get_object("treeview4"),
                      self.get_object("treeview5"),
                      self.get_object("treeview6"),
                      self.get_object("treeview7"),
                      self.get_object("treeview8"),
                      self.get_object("treeview9"),
                      self.get_object("treeview10"),
                      self.get_object("treeview11"),
                      self.get_object("treeview12")):
            self.clear_table(table)
        self.update_tables()
    def resize_click(self,widget,data=None):
        for table in (self.get_object("treeview3"),
                      self.get_object("treeview4"),
                      self.get_object("treeview5"),
                      self.get_object("treeview6"),
                      self.get_object("treeview7"),
                      self.get_object("treeview8"),
                      self.get_object("treeview9"),
                      self.get_object("treeview10"),
                      self.get_object("treeview11"),
```

```
                self.get_object("treeview12")):
            self.clear_table(table)
        n=round(self.get_object("adjustment1").get_value())
        m=round(self.get_object("adjustment2").get_value())
        for table in ("treeview1",
                      "treeview2"):
            mat=[[0.0 for j in range(0,m)] for i in range(0,n)]
            model=self.get_object(table).get_model()
            i=0;j=0;
            for x in model:
                i+=1;j=0
                if i>n:
                    break;
                for y in x:
                    j+=1
                    if j>m:
                        break
                    mat[i-1][j-1]=y

            self.clear_table(self.get_object(table))
            self.create_table(table,2,str,mat,True)
        mat=[[str(0.0) for j in range(0,m)] for i in range(0,n)]
        self.create_table("treeview3",2,str,mat)
        row_model=self.get_object("liststoreRows")
        row_model.clear()
        for i in range(1,n+1):
            row_model.append([i])
    def update_tables(self):
        n=round(self.get_object("adjustment1").get_value())
        m=round(self.get_object("adjustment2").get_value())
        empty_mat=[[str(0.0) for j in range(0,m)] for i in range(0,n)]
        self.create_table("treeview1",2,str,empty_mat,True)
        self.create_table("treeview2",2,str,empty_mat,True)
        self.create_table("treeview3",2,str,empty_mat)
        row_model=self.get_object("liststoreRows")
        row_model.clear()
        for i in range(1,n+1):
            row_model.append([i])
    def on_window_destroy( self,widget, data=None):
        self.get_object('window1').hide()
        Gtk.main_quit()
    def create_table(self,name,dim,el_type,data,editable=False):
        table=self.get_object(name)
        model=table.get_model()
        self.clear_table(table)
        if dim==1:
            m=len(data)
        elif dim==2:
            m=len(data[0])
        model=Gtk.ListStore(*(el_type for j in range(0,m)))
        if dim==1:
            model.append(data)
        elif dim==2:
            for i in data:
                model.append(i)
        table.set_model(model)
        for i in range(0,m):
            rend = Gtk.CellRendererText()
            if editable:
```

```
                rend.set_property("editable",True)
                if el_type==str:
                    rend.connect("edited",self.on_cell_edit_f,(model,i))
            col = Gtk.TreeViewColumn(str(i+1), rend)
            col.add_attribute(rend,"text",i)
            table.append_column(col)
    def on_cell_edit_f(self, widget, path, text,data=None):
        model,col=data
        model[path][col]=str(float(text))
    def on_matrix_a_ok(self,widget,data=None):
        a=[[float(j) for j in i] for i in self.get_object("treeview1").get_model()]
        rrcu_a=RRCU(a)
        self.create_table("treeview4",1,int,rrcu_a.rows_ptrs)
        self.create_table("treeview5",1,int,rrcu_a.cols_nums)
        self.create_table("treeview6",1,float,rrcu_a.elements)
        self.rrcu_a=rrcu_a

    def on_matrix_b_ok(self,widget,data=None):
        b=[[float(j) for j in i] for i in self.get_object("treeview2").get_model()]
        rrcu_b=RRCU(b)
        self.create_table("treeview7",1,int,rrcu_b.rows_ptrs)
        self.create_table("treeview8",1,int,rrcu_b.cols_nums)
        self.create_table("treeview9",1,float,rrcu_b.elements)
        self.rrcu_b=rrcu_b

    def input_data(self):
        try:
            if self.rrcu_a==None:
                print(1)
                self.show_msg("Матрица А не создана!")
                raise
        except:
            self.show_msg("Матрица А не создана!")
            raise
        try :
            if self.rrcu_b==None:
                self.show_msg("Матрица В не создана!")
                raise
        except:
            self.show_msg("Матрица В не создана!")
            raise
        return self.rrcu_a,self.rrcu_b
    def output_data(self,rrcu_c):
        if rrcu_c:
            self.create_table("treeview10",1,int,rrcu_c.rows_ptrs)
            self.create_table("treeview11",1,int,rrcu_c.cols_nums)
            self.create_table("treeview12",1,float,rrcu_c.elements)
            n=round(self.get_object("adjustment1").get_value())
            m=round(self.get_object("adjustment2").get_value())
            c=[[str(j) for j in i] for i in expand_rrcu(rrcu_c,m)]
            self.create_table("treeview3",2,str,c)
    def on_run_click(self,widget,data=None):
        rrcu_a,rrcu_b=self.input_data()
        rrcu_c=sum_rrcu(rrcu_a,rrcu_b);
        self.output_data(rrcu_c);

app=Application("prac2.ui")
app.show("window1")
```