ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

КОНСОЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ НАХОЖДЕНИЯ МАТРИЦЫ АЛГЕБРАИЧЕСКИХ ДОПОЛНЕНИЙ

Пояснительная записка

Исполнитель:
студент группы БПИ199
_____ К. Н. Борисов
«_____» _____ 2020 г.

Москва 2020

Содержание

1. Текст задания

Найти алгебраическое дополнение для каждого элемента матрицы. Входные данные: целое положительное число n, произвольная матрица A размерности $n \times n$. Количество потоков является входным параметром, при этом размерность матриц может быть не кратна количеству потоков. [1]

2. Применяемые расчетные методы

2.1. Алгоритм вычисления

Алгебраическое дополнение это $A_{ij} = (-1)^{i+j} M_{ij}$, где M_{ij} – дополнительный минор, определитель матрицы, получающейся из исходной матрицы A путём вычёркивания i-й строки и j-го столбца. [3] Для вычисления определителя использовался рекурсивный алгоритм $\Delta = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} M_j^1$. [4] Именно поэтому программа не работает на матрицах больше чем 10×10.

2.2. Распределение по потокам

В идеале у нас бы было по одному потоку на каждый элемент матрицы, но у нам нужно использовать заданное число потоков. Поэтому каждый поток получает некоторое количество идущих подряд элементов матрицы. Мы пытаемся распределить поровну, но у нас размерность матриц может быть не кратна количеству потоков. Поэтому приходится распределять почти поровну, используя алгоритм чем-то напоминающий распределение високосных лет. Получается что в программе использовался итеративный параллелизм.

2.3. Вывод результата в консоль

В консоль матрицы выводятся в JSON-подобном формате, чтобы их можно было сразу вставить в питон и проверить правильность результата.

2.4. Входные данные

На вход в аргументах командной строки (`argv`) подаются количество потоков и имя входного файла с матрицей. Имя входного файла опционально: если его нет то матрица будет читаться из консоли (`stdin`). Файл с матрицей имеет следующий формат: на перовой строке два целых числа – размеры матрицы n и m, потом идут $n \times m$ действительных чисел, разделённых пробельными символами (`'\n'` или `' '`).

3. Тестовые примеры

Программа корректно работает с большими матрицами. (см. рисунок 1)

```
cortan122@ThinkPad-T480: /mnt/c/Users/user/OneDrive/dz2020/asm/task03
cortan122@ThinkPad-T480:/mnt/c/Users/user/OneDrive/dz2020/asm/task03$ time ./matrix 10 test100.txt
Inputted matrix:
[
  [         0,          3,          1,          4,          1,          4,          4,          1,          0,          4, ],
  [         0,          0,          3,          1,          0,          4,          1,          0,          1,          1, ],
  [         0,          3,          1,          3,          2,          2,          3,          1,          0,          1, ],
  [         1,          0,          2,          0,          2,          1,          1,          1,          2,          0, ],
  [         1,          3,          0,          0,          1,          1,          0,          1,          1,          0, ],
  [         1,          2,          1,          0,          1,          1,          1,          4,          1,          1, ],
  [         0,          3,          0,          1,          2,          4,          3,          0,          1,          0, ],
  [         1,          1,          0,          1,          2,          0,          1,          1,          4,          1, ],
  [         1,          0,          3,          1,          2,          1,          1,          1,          0,          4, ],
  [         1,          1,          1,          0,          1,          3,          1,          2,          2,          4, ],
]

starting thread from 0 to 10
starting thread from 10 to 20
starting thread from 20 to 30
starting thread from 30 to 40
starting thread from 40 to 50
starting thread from 50 to 60
starting thread from 60 to 70
starting thread from 70 to 80
starting thread from 80 to 90
starting thread from 90 to 100
Resulting matrix:
[
  [      2184,        405,        468,       -624,      -1980,       -786,       2127,       -765,        132,        105, ],
  [     -2212,       1190,       1540,      -1064,      -1036,       -798,        616,       -434,        910,        336, ],
  [      1148,      -3544,      -3782,       5290,       5042,       3888,      -6092,       2572,      -2018,      -1386, ],
  [      6650,      -3811,      -3398,       4036,       2390,       3186,      -2663,       1345,      -2138,      -1785, ],
  [      2030,        194,       -254,        586,       -124,        360,       -674,       -284,       -338,       -252, ],
  [     -2842,       2336,       2518,      -3110,      -2434,      -2412,       3106,       -620,       1300,        798, ],
  [     -3388,       2672,       2560,      -3908,      -2098,      -2454,       3568,      -1502,       1426,       1050, ],
  [     -3206,       2242,       2228,      -2476,      -1892,      -2334,       2354,      -1102,       1808,        966, ],
  [     -3374,       2908,       3146,      -3700,      -2180,      -2934,       3230,      -1618,       1382,       1386, ],
  [      2968,      -3869,      -4240,       4664,       4240,       4134,      -5353,       2279,      -2014,      -1113, ],
]

real    0m0.665s
user    0m4.578s
sys     0m0.000s
cortan122@ThinkPad-T480:/mnt/c/Users/user/OneDrive/dz2020/asm/task03$
```

Рисунок 1: Большая матрица

Программа работает с большими матрицами быстрее, когда у неё несколько потоков, получается 0.6сек и 2.3сек. (см. рисунок 2)

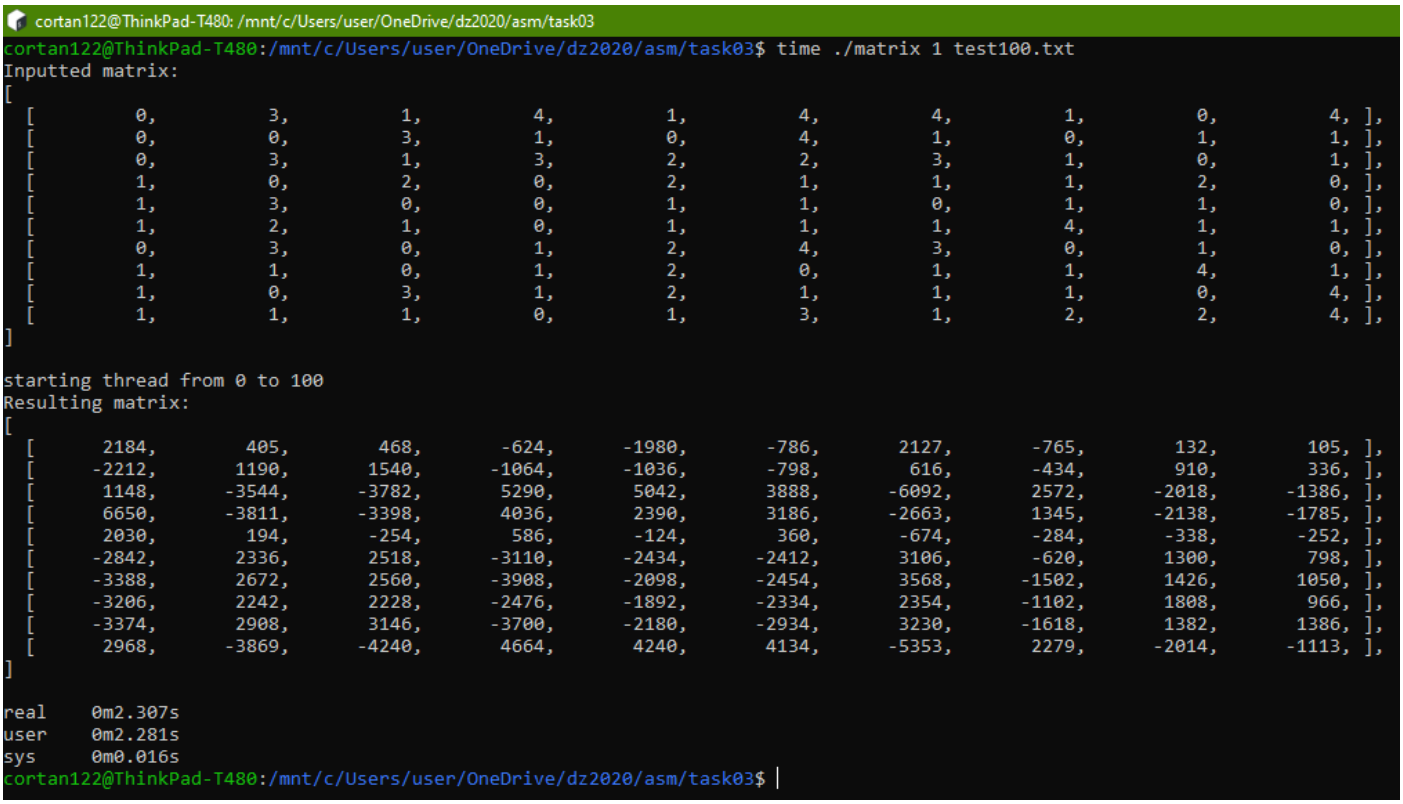

Рисунок 2: Большая матрица с одним потоком

Программа корректно работает когда размерность матрицы не кратна количеству потоков $9 \bmod 7 \neq 0$. (см. рисунок 3)

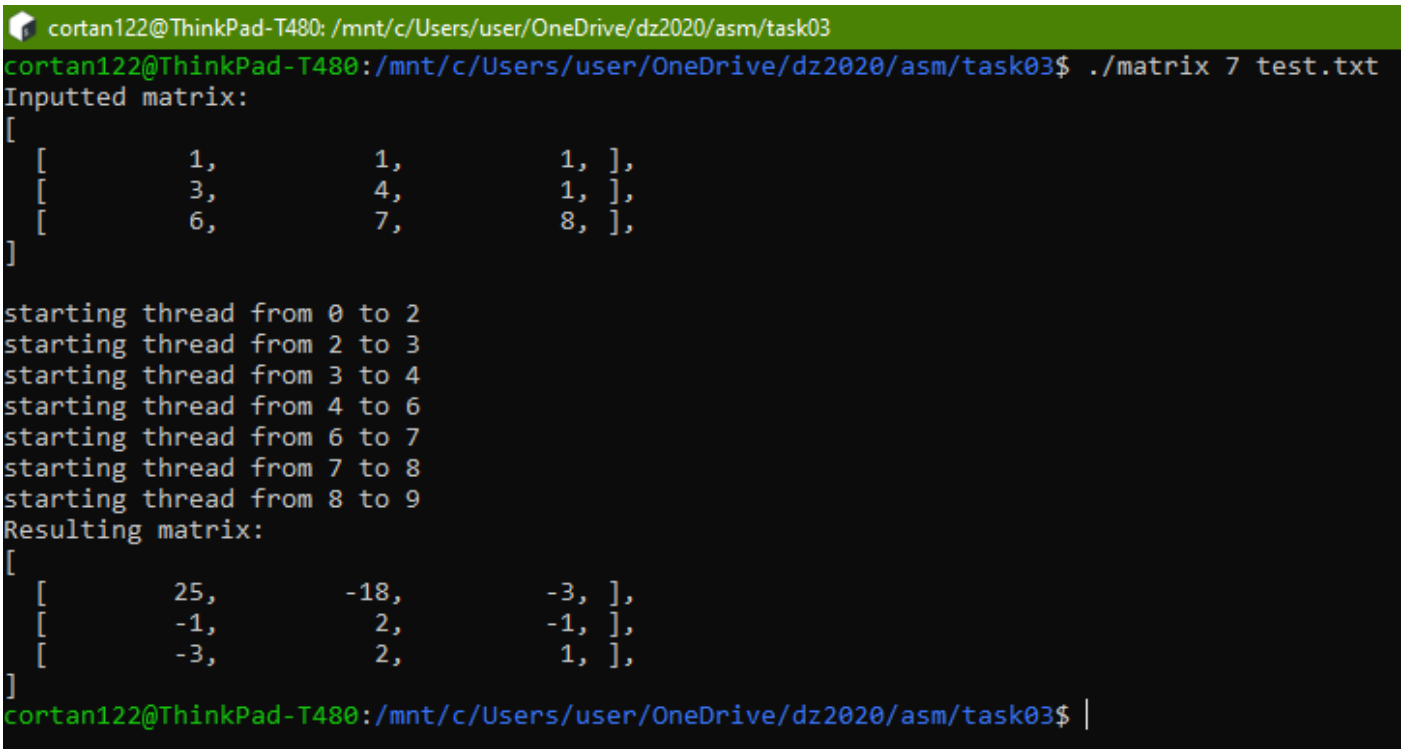

Рисунок 3: Неровное количество потоков

В программе отсутствуют утечки памяти. (см. рисунок 4)

```
cortan122@ThinkPad-T480: /mnt/c/Users/user/OneDrive/dz2020/asm/task03
cortan122@ThinkPad-T480:/mnt/c/Users/user/OneDrive/dz2020/asm/task03$ valgrind ./matrix 9 test.txt
==171== Memcheck, a memory error detector
==171== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==171== Using Valgrind-3.16.1 and LibVEX; rerun with -h for copyright info
==171== Command: ./matrix 9 test.txt
==171==
==171== error calling PR_SET_PTRACER, vgdb might block
Inputted matrix:
[
  [         1,          1,          1, ],
  [         3,          4,          1, ],
  [         6,          7,          8, ],
]

starting thread from 0 to 1
starting thread from 1 to 2
starting thread from 2 to 3
starting thread from 3 to 4
starting thread from 4 to 5
starting thread from 5 to 6
starting thread from 6 to 7
starting thread from 7 to 8
starting thread from 8 to 9
Resulting matrix:
[
  [        25,        -18,         -3, ],
  [        -1,          2,         -1, ],
  [        -3,          2,          1, ],
]
==171==
==171== HEAP SUMMARY:
==171==     in use at exit: 0 bytes in 0 blocks
==171==   total heap usage: 33 allocs, 33 frees, 11,832 bytes allocated
==171==
==171== All heap blocks were freed -- no leaks are possible
==171==
==171== For lists of detected and suppressed errors, rerun with: -s
==171== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
cortan122@ThinkPad-T480:/mnt/c/Users/user/OneDrive/dz2020/asm/task03$
```

Рисунок 4: Отсутствие утечек памяти

4. Список использованной литературы

[1] Практические приемы построения многопоточных приложений [Электронный ресурс]. //URL: http://softcraft.ru/edu/comparch/tasks/t03/ (Дата обращения: 15.11.2020, режим доступа: свободный)

[2] pthreads(7) Linux User's Manual [Электронный ресурс]. //URL: https://man7.org/linux/man-pages/man7/pthreads.7.html (Дата обращения: 15.11.2020, режим доступа: свободный)

[3] Статья «Алгебраическое дополнение» Wikipedia.org //URL: https://ru.wikipedia.org/wiki/Алгебраическое_дополнение (Дата обращения: 15.11.2020, режим доступа: свободный)

[4] Статья «Определитель» Wikipedia.org //URL: https://ru.wikipedia.org/wiki/Определитель (Дата обращения: 15.11.2020, режим доступа: свободный)

5. Текст программы

```
#include <stdlib.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdio.h>
#include <time.h>
#include <assert.h>
#include <pthread.h>

typedef struct Matrix {
  int n;
  int m;
  double* mat;
} Matrix;

typedef struct MatrixThreadShuttle {
  Matrix* in;
  Matrix* out;
  int startIndex;
  int endIndex;
} MatrixThreadShuttle;

Matrix makeMatrix(int n, int m){
  Matrix r;
  r.n = n;
  r.m = m;
  r.mat = calloc(n*m*sizeof(double), 1);
  return r;
}

// печатает матрицу в JSON-подобном формате
void printMatrix(Matrix* m){
  printf("[\n");
  for(int i = 0; i < m->n; i++){
    printf("  [");
    for(int j = 0; j < m->m; j++){
      printf("%10.6g, ", m->mat[i*m->m + j]);
    }
    printf("],\n");
  }
  printf("]\n");
}

// находит минормую матрицу (делает копию)
Matrix minorMatrix(Matrix* m, int i_, int j_){
  Matrix matrix = makeMatrix(m->n - 1, m->n - 1);

  int i1 = 0;
  for(int i = 0; i < m->n; i++){
    if(i == i_)continue;
    int j1 = 0;
    for(int j = 0; j < m->m; j++){
      if(j == j_)continue;
      matrix.mat[i1*matrix.m + j1] = m->mat[i*m->m + j];
      j1++;
    }
    i1++;
  }

  return matrix;
}

// тут надо объявлять хедер для рекурсии
```

```
double det(Matrix* m);

// находит дополнительный минор
double minorDet(Matrix* m, int i, int j){
  Matrix t = minorMatrix(m, i, j);
  double r = det(&t);
  free(t.mat);
  return r;
}

// рекурсивно считаем определитель
double det(Matrix* m){
  assert(m->m == m->n);
  assert(m->n > 1);
  if(m->n == 2){
    return m->mat[0]*m->mat[3] - m->mat[1]*m->mat[2];
  }

  double r = 0;
  int delta = 1;
  for(int i = 0; i < m->m; i++){
    r += m->mat[i] * minorDet(m, 0, i) * delta;
    delta *= -1;
  }
  return r;
}

// находит алгебраическое дополнение
double adjunct(Matrix* m, int i, int j){
  return minorDet(m, i, j) * ((i+j) % 2? -1 : 1);
}

// считывает матрицу из файла
Matrix readMatrix(char* filename){
  FILE* file = fopen(filename, "r");
  if(!file){
    perror(filename);
    exit(1);
  }

  int n,m;
  fscanf(file, "%d %d", &n, &m);
  Matrix matrix = makeMatrix(n,m);
  for(int i = 0; i < n*m; i++){
    fscanf(file, "%lf", matrix.mat + i);
  }

  fclose(file);
  return matrix;
}

void* threadFunction(void* mts){
  MatrixThreadShuttle* args = mts;

  for(int i = args->startIndex; i < args->endIndex; i++){
    args->out->mat[i] = adjunct(args->in, i / args->in->m, i % args->in->m);
  }

  free(args);
  return NULL;
}

// тут мы просто упаковываем все аргументы в поток в void*
pthread_t makeThread(Matrix* in, Matrix* out, int i, int j){
```

```
  printf("starting thread from %d to %d\n", i, j);

  MatrixThreadShuttle* arg = malloc(sizeof(MatrixThreadShuttle));
  arg->startIndex = i;
  arg->endIndex = j;
  arg->in = in;
  arg->out = out;
  pthread_t res;
  pthread_create(&res, NULL, threadFunction, arg);
  return res;
}

pthread_t* makeThreads(int n, int m, int threadCount, Matrix* in, Matrix* out){
  pthread_t* threads = malloc(threadCount*sizeof(pthread_t));
  int threadIndex = 0;

  // тут мы пытаемся поровну распределить елементы матрицы по потокам
  double delta = n*m/(double)threadCount;
  double acc = 0;
  int i = -1;
  for(int j = 0; j < n*m; j++){
    acc += 1;
    if(acc >= delta){
      acc -= delta;
      threads[threadIndex++] = makeThread(in, out, i+1, j+1);
      i = j;
    }
  }
  // тут нам надо в конце проверять использовали ли мы все елементы
  if(i+1 != n*m){
    threads[threadIndex++] = makeThread(in, out, i+1, n*m);
  }

  return threads;
}

int main(int argc, char** argv){
  if(argc == 1){
    // формат командный строки для тех кому лень читать ПЗ
    fprintf(stderr, "./matrix numberOfThreads [infile]\n");
    return 1;
  }

  int threadCount = atoi(argv[1]);

  char* infile = "/dev/fd/0";
  if(argc > 2){
    infile = argv[2];
  }else{
    fprintf(stderr, "readring stdin...\n");
  }

  Matrix matrix = readMatrix(infile);
  if(matrix.n != matrix.m){
    fprintf(stderr, "we need a square matrix\n");
    free(matrix.mat);
    return 1;
  }else if(matrix.n > 10){
    fprintf(stderr, "this matrix is SO big.... and i think i cant handle it\n");
    free(matrix.mat);
    return 1;
  }
  Matrix outmatrix = makeMatrix(matrix.n, matrix.m);
```

```
  printf("Inputted matrix:\n");
  printMatrix(&matrix);
  printf("\n");

  if(threadCount > matrix.n*matrix.m){
    printf("we have a little too many threads; we cant use all of them a once\n");
    threadCount = matrix.n*matrix.m;
  }

  pthread_t* threads = makeThreads(matrix.n, matrix.m, threadCount, &matrix, &outmatrix);
  for(int i = 0; i < threadCount; i++){
    pthread_join(threads[i], NULL);
  }
  free(threads);

  printf("Resulting matrix:\n");
  printMatrix(&outmatrix);

  free(matrix.mat);
  free(outmatrix.mat);
  return 0;
}
```