Белорусский государственный технологический университет

факультет информационных технологий

кафедра информационных систем и технологий

**Отчёт по предмету**

**«Математическое программирование»**

**Вариант 12**

Немкович Анастасия Вадимовна

ФИТ, 2 курс, группа 1/1

Минск 2023

**ЦЕЛЬ РАБОТЫ:** освоить общие принципы решения задач методом динамического программирования, сравнить полученные решения задач с рекурсивным методом.

**ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ:**

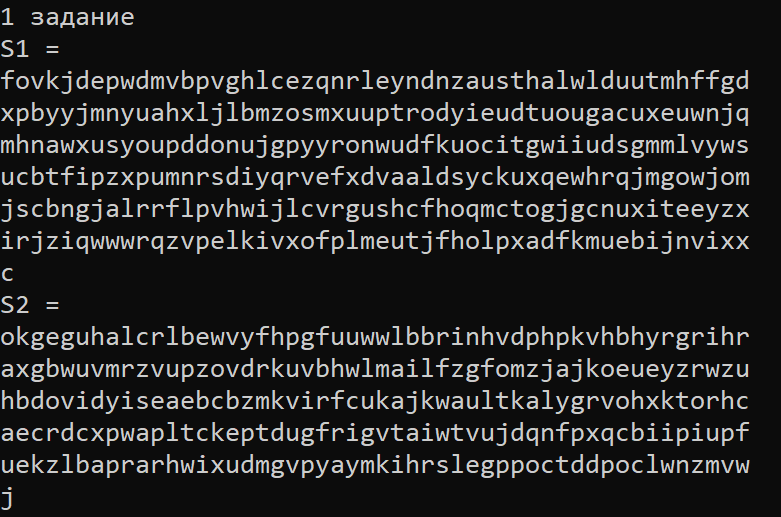
***Задание 1.***

На языке С++ сгенерировать случайным образом строку букв латинского алфавита S1 длиной 300 символов и S2 длиной 200.

Код программы:

|  |
| --- |
| cout << "1 задание" << endl;  srand(time(NULL));  char abc[25];  char s1[300];  char s2[250];  for (int i = 97, n = 0; i <= 122; ++i, ++n)  {  abc[n] = (char)i;  }  cout << "S1 = ";  for (int i = 0; i < 301; i++)  {  s1[i] = abc[\_rand(0, 25)];  if (i % 50 == 0)  cout << "\n";  cout << s1[i];  }  cout << "\nS2 =";  for (int i = 0; i < 251; i++)  {  s2[i] = abc[\_rand(0, 25)];  if (i % 50 == 0)  cout << "\n";  cout << s2[i];  }  cout << "\n"; |

Результат выполнения:



***Задание 2.***

Вычислить двумя способами (рекурсивно и с помощью динамического программирования)  – дистанцию Левенштейна для , где - длина строки ,  - строка состоящая из первых  символов строки . (копии экрана и код вставить в отчет).

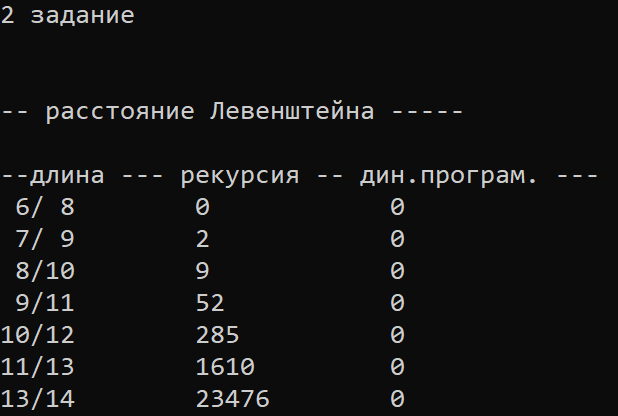
Код реализации через динамическое программирование:

|  |
| --- |
| int levenshtein(int lx, const char x[], int ly, const char y[])  {  int\* d = new int[(lx + 1) \* (ly + 1)];  for (int i = 0; i <= lx; i++) DD(i, 0) = i;  for (int j = 0; j <= ly; j++) DD(0, j) = j;  for (int i = 1; i <= lx; i++)  for (int j = 1; j <= ly; j++)  {  DD(i, j) = min3(DD(i - 1, j) + 1, DD(i, j - 1) + 1,  DD(i - 1, j - 1) + (x[i - 1] == y[j - 1] ? 0 : 1));  }  return DD(lx, ly);  } |

Реализации рекурсивным методом:

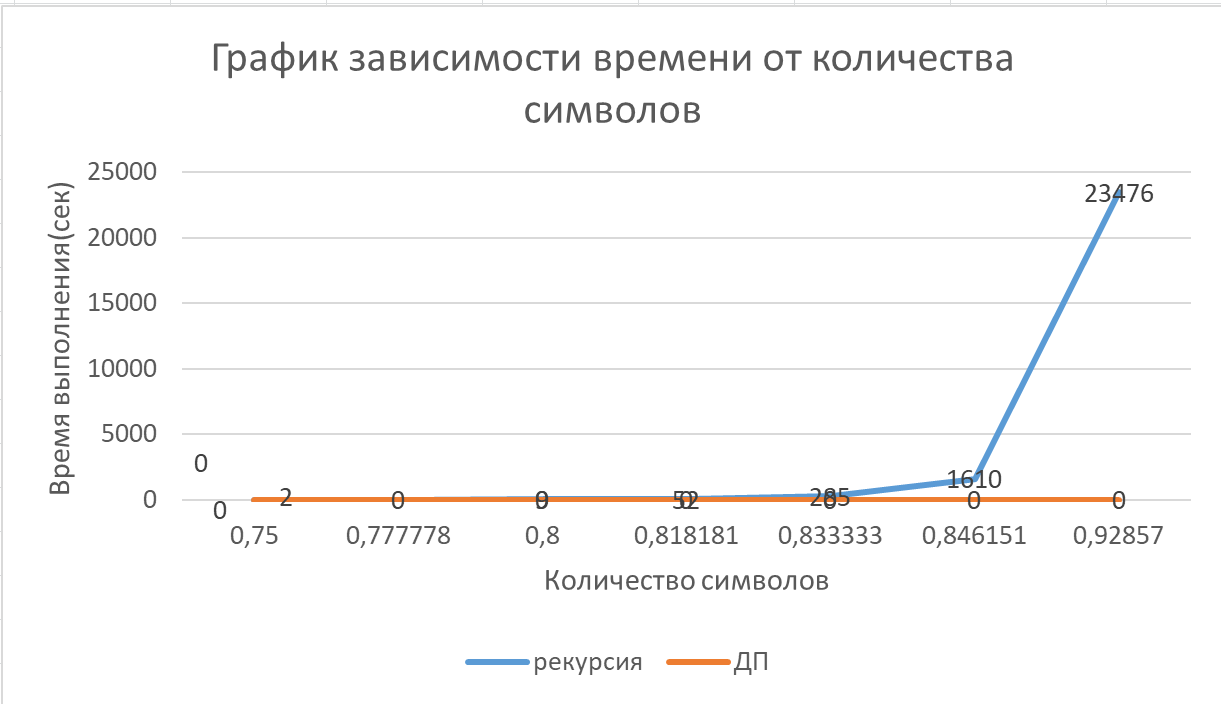
|  |
| --- |
| int levenshtein\_r(  int lx, const char x[],  int ly, const char y[]  )  {  int rc = 0;  if (lx == 0) rc = ly;  else if (ly == 0) rc = lx;  else if (lx == 1 && ly == 1 && x[0] == y[0]) rc = 0;  else if (lx == 1 && ly == 1 && x[0] != y[0]) rc = 1;  else rc = min3(  levenshtein\_r(lx - 1, x, ly, y) + 1,  levenshtein\_r(lx, x, ly - 1, y) + 1,  levenshtein\_r(lx - 1, x, ly - 1, y) + (x[lx - 1] == y[ly - 1] ? 0 : 1)  );  return rc;  }; |

Результат выполнения:



***Задание 3.***

Выполнить сравнительный анализ времени затраченного на вычисление дистанции Левенштейна для двух методов решения. Построить графики зависимости времени вычисления от . (копии экрана и график вставить в отчет).



***Задание 4.***

Реализовать вручную пример вычисления дистанции Левенштейна при помощи рекурсивного алгоритма (в соответствии с вариантом) (каждый шаг алгоритма по примеру из лекции вставить в отчет).

|  |  |
| --- | --- |
| Вариант 12 | |
| Эхо | Хорек |

1. **Эхо, Хорек**

min(Эх, Хорек+1)

min ( Эхо, Хоре+1)

min (Эх, Хоре+1)

1. **Эх, Хорек**

min(Э, Хорек+1)

min (Эх, Хоре+1)

min (Э, Хоре+1)

1. **Эхо, Хоре**

min(Эх, Хоре +1)

min (Эхо, Хор+1)

min (Эх, Хор +1)

1. **Эх, Хоре**

min(Э, Хоре +1)

min (Эх, Хор +1)

min (Э, Хор+1)

1. **Э, Хорек**

min(« », Хорек) = 5

min (Э, Хоре+1)

min («»,Хоре) = 4

1. **Э, Хоре**

min («»,Хоре) = 4

min (Э, Хор+1)

min («»,Хор) = 3

1. **Эхо, Хор**

min(Эх, Хор +1)

min (Эхо, Хо +1)

min (Эх, Хо +1)

1. **Эх, Хор**

min(Э, Хор +1)

min (Эх, Хо+1)

min (Э, Хо+1)

1. **Э, Хор**

min(«»,Хор) = 3

min (Э, Хо+1)

min («»,Хо) = 2

1. **Эхо, Хо**

min(Эх, Хо+1)

min (Эхо, Х+1)

min (Эх, Х+1)

1. **Эх, Хо**

min(Э, Хо+1)

min (Эх, Х+1)

min (Э, Х)

1. **Э, Хо**

min(«», Хо) = 2

min (Э, Х+1)

min («», Х) = 1

1. **Эхо, Х**

min(Эх, Х+1)

min (Эхо, «») = 3

min (Эх, «») = 2

1. **Эх, Х**

min(Э, Х+1)

min (Эх, «») = 2

min (Х, «») = 1

1. **Э, Х**

min(«», Х) = 1

min (Э, «») = 1

min («», «») =0

Э, Х = min (2,2,1) = 1

Эх, Х = min (3,2,1) = 1

Эхо, Х = min (3,3,2) = 2

Э, Хо = min (2,3,1) = 1

Эх, Хо = min (2,2,1) = 1

Эхо, Хо = min (2,2,1) = 1

Э, Хор = min (3,2,2) = 2

Эх, Хор = min (3,2,2) = 2

Эхо, Хор = min (3,2,2) = 2

Э, Хоре = min (4,3,3) = 3

Э, Хорек = min (5,4,4) = 4

Эх, Хоре = min (4,3,3) = 3

Эхо, Хоре = min (4,3,3) = 3

Эх, Хорек = min (5,4,4) = 4

Эхо, Хорек = min (5,4,4) = 4

***Задание 5.***

**Четные варианты**. Выполнить сравнительный анализ времени затраченного на решение задачи об оптимальной расстановке скобок при умножении нескольких матриц для двух методов решения (рекурсивное решение, динамическое программирование). Размерность матриц взять в соответствии с вариантом. Объяснить в отчете принцип расставления скобок по итоговой матрице + код + копии экрана.

|  |
| --- |
| 8\*11, 11\*19, 19\*22, \*29, 29\*39, 39\*50 |

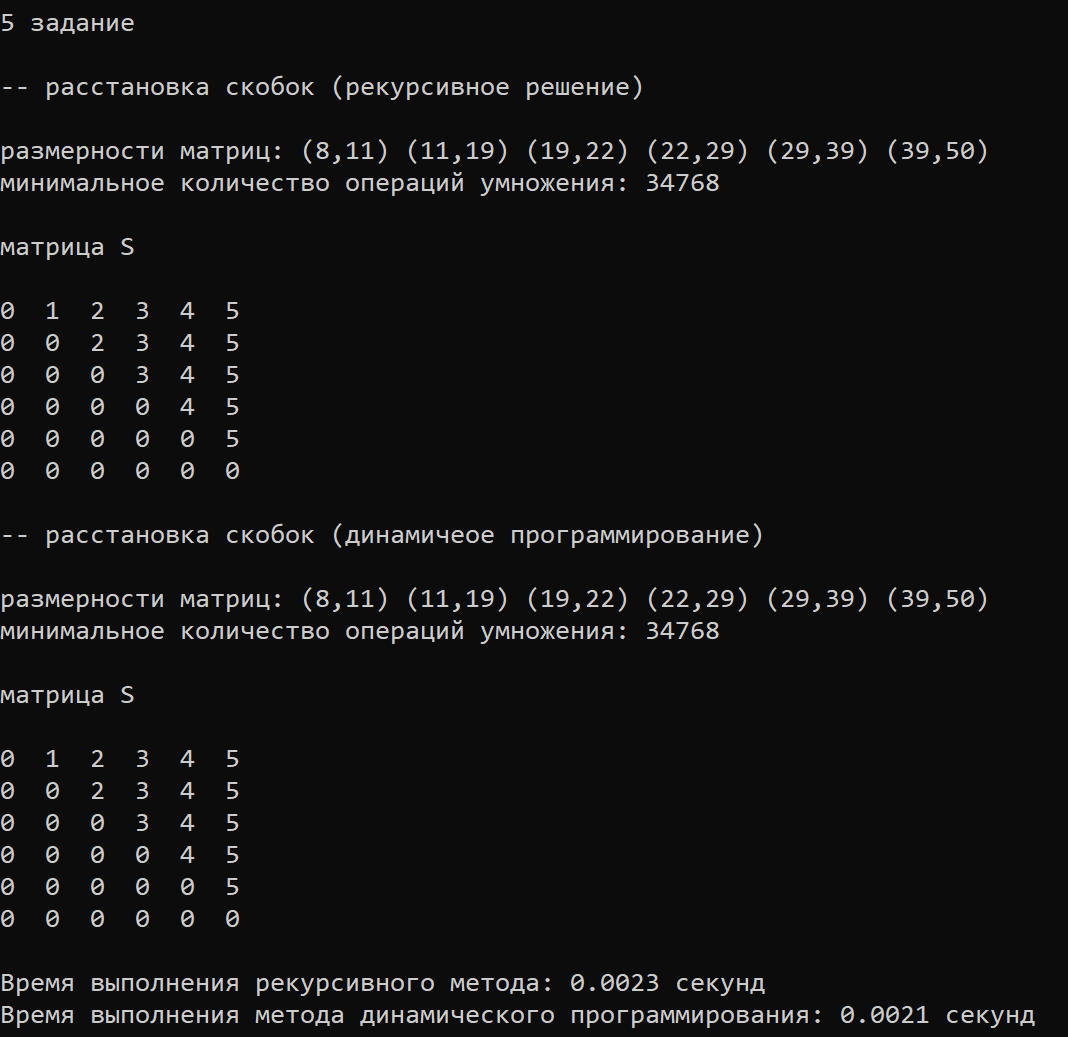
Код программы:

|  |
| --- |
| // MultiMatrix.cpp  #include "stdafx.h"  #include <memory.h>  // расстановка скобок (рекурсия)  #define INFINITY 0x7fffffff  #define NINFINITY 0x80000000  int OptimalM(int i, int j, int n, const int c[], int\* s)  {  #define OPTIMALM\_S(x1,x2) (s[(x1-1)\*n+x2-1])  int o = INFINITY;  int bo = INFINITY;  if (i < j)  {  for (int k = i; k < j; k++)  {  bo = OptimalM(i, k, n, c, s) + OptimalM(k + 1, j, n, c, s) + c[i - 1] \* c[k] \* c[j];  if (bo < o)  {  o = bo;  OPTIMALM\_S(i, j) = k;  }  }  }  else o = 0;  return o;  #undef OPTIMALM\_S  };  // расстановка скобок (динамическое программирование)  int OptimalMD(int n, const int c[], int\* s)  {  #define OPTIMALM\_S(x1,x2) (s[(x1-1)\*n+x2-1])  #define OPTIMALM\_M(x1,x2) (M[(x1-1)\*n+x2-1])  int\* M = new int[n \* n], j = 0, q = 0;  for (int i = 1; i <= n; i++)  OPTIMALM\_M(i, i) = 0;  for (int l = 2; l <= n; l++)  {  for (int i = 1; i <= n - l + 1; i++)  {  j = i + l - 1;  OPTIMALM\_M(i, j) = INFINITY;  for (int k = i; k <= j - 1; k++)  {  q = OPTIMALM\_M(i, k) + OPTIMALM\_M(k + 1, j) + c[i - 1] \* c[k] \* c[j];  if (q < OPTIMALM\_M(i, j))  {  OPTIMALM\_M(i, j) = q; OPTIMALM\_S(i, j) = k;  }  }  }  }  return OPTIMALM\_M(1, n);  #undef OPTIMALM\_M  #undef OPTIMALM\_S  }; |

|  |
| --- |
| //MultyMatrix.h  #pragma once  // расстановка скобок при умножении матриц  // функции возвращают минимальное количество операций умножения  #define OPTIMALM\_PARM(x) ((int\*)x) // для представления 2мерного массива  int OptimalM( // рекурсия  int i, // [in] номер первой матрицы  int j, // [in] номер последней матрицы  int n, // [in] количество матриц  const int c[], // [in] массив размерностей  int\* s // [out] результат: позиции скобок  );  int OptimalMD( // динамическое программирование  int n, // [in] количество матриц  const int c[], // [in] массив размерностей  int\* s // [out] результат: позиции скобок  ); |

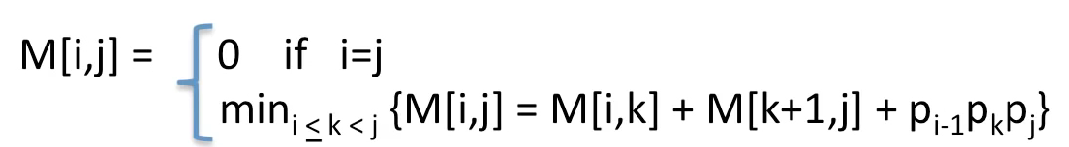
|  |
| --- |
| //main  #include "MultyMatrix.h"  #include <chrono>  #define N 6  // Функция для замера времени выполнения метода OptimalM  double measureTimeOptimalM(const int\* Mc) {  auto start = std::chrono::high\_resolution\_clock::now(); // Замер времени начала выполнения  int Ms[N][N];  int r = OptimalM(1, N, N, Mc, OPTIMALM\_PARM(Ms)); // Вызов рекурсивной функции  auto end = std::chrono::high\_resolution\_clock::now(); // Замер времени окончания выполнения  std::chrono::duration<double> elapsed = end - start; // Вычисление времени выполнения в секундах  return elapsed.count() \* 1000; // Возвращаем время в миллисекундах  }  // Функция для замера времени выполнения метода OptimalMD  double measureTimeOptimalMD(const int\* Mc) {  auto start = std::chrono::high\_resolution\_clock::now(); // Замер времени начала выполнения  int Ms[N][N];  int rd = OptimalMD(N, Mc, OPTIMALM\_PARM(Ms)); // Вызов функции динамического программирования  auto end = std::chrono::high\_resolution\_clock::now(); // Замер времени окончания выполнения  std::chrono::duration<double> elapsed = end - start; // Вычисление времени выполнения в секундах  return elapsed.count() \* 1000; // Возвращаем время в миллисекундах  }  cout << "5 задание" << endl;  int Mc[N + 1] = { 8, 11, 19, 22, 29, 39, 50 }, Ms[N][N], r = 0, rd = 0;  memset(Ms, 0, sizeof(int) \* N \* N);  r = OptimalM(1, N, N, Mc, OPTIMALM\_PARM(Ms));  setlocale(LC\_ALL, "rus");  cout << endl << "-- расстановка скобок (рекурсивное решение) "  << endl;  cout << endl << "размерности матриц: ";  for (int i = 1; i <= N; i++) cout << "(" << Mc[i - 1] << "," << Mc[i] << ") ";  cout << endl << "минимальное количество операций умножения: " << r;  cout << endl << endl << "матрица S" << endl;  for (int i = 0; i < N; i++)  {  cout << endl;  for (int j = 0; j < N; j++) cout << Ms[i][j] << " ";  }  cout << endl;  memset(Ms, 0, sizeof(int) \* N \* N);  rd = OptimalMD(N, Mc, OPTIMALM\_PARM(Ms));  cout << endl  << "-- расстановка скобок (динамичеое программирование) " << endl;  cout << endl << "размерности матриц: ";  for (int i = 1; i <= N; i++)  cout << "(" << Mc[i - 1] << "," << Mc[i] << ") ";  cout << endl << "минимальное количество операций умножения: "  << rd;  cout << endl << endl << "матрица S" << endl;  for (int i = 0; i < N; i++)  {  cout << endl;  for (int j = 0; j < N; j++) cout << Ms[i][j] << " ";  }  cout << endl << endl;  double timeRecursive = measureTimeOptimalM(Mc);  std::cout << "Время выполнения рекурсивного метода: " << timeRecursive << " секунд" << std::endl;  // Измерение времени для метода динамического программирования  double timeDynamic = measureTimeOptimalMD(Mc);  std::cout << "Время выполнения метода динамического программирования: " << timeDynamic << " секунд" << std::endl; |

Результат выполнения:



8\*11, 11\*19, 19\*22, \*29, 29\*39, 39\*50

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i/j | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 0 | 1 672 | 5016 | 10120 | 19168 | 34768 |
| 2 | x | 0 | 4 598 | 11616 | 24057 | 45507 |
| 3 | x | X | 0 | 12 122 | 33611 | 70661 |
| 4 | X | x | x | 0 | 24 882 | 67782 |
| 5 | x | x | x | X | 0 | 56 550 |
| 6 | x | x | x | x | x | 0 |



P0 = 8

P1 = 11

P2 = 19

P3 = 22

P4 = 29

P5 = 39

P6 = 50

Первая диагональ

M(1,2) = (0+0+8\*11\*19) = 1 672

M(2,3) = (0+0+ 11\*19\*22) = 4 598

M(3,4) = (0+0+ 19\*22\*29) = 12 122

M(4,5) = (0+0+ 22\*29\*39) = 24 882

M(5,6) = (0+0+ 29\*39\*50) = 56 550

M(1,1) = M(2,2) = M(3,3) = M(4,4) = M(5,5) = M(6,6)

Вторая диагональ

М(1,3)

K=1 : M(1,1) + M(2,3) + p0p1p3 = 0 + 4 598 + 8\*11\*22 = 6534

K = 2 : M(1,2) + M(3,3) + p0p2p3 = 1 672 + 0 + 8\*19\*22 = 5016

M(2,4)

K = 2: M(2,2) + M(3,4) + p1p2p4 = 0 + 12 122 + 11\*19\*29 = 18183

K = 3: M(2,3) + M(4,4) + p1p3p4 = 4 598 + 0 +11\*22\*29 = 11616

M(3,5)

K = 3: M(3,3) + M(4,5) + p2p3p5 = 0 + 24 882 + 19\*22\*39 = 41184

K = 4 : M(3,4) + M(5,5) + p2p4p5 = 12 122 + 0 + 19\*29\*39 = 33611

M(4,6)

K = 4 : M(4,4) + M(5,6) + p3p4p6 = 0 + 56 550+ 22\*29\*50 = 88450

K = 5 : M(4,5) + M(6,6) + p3p5p6 = 24 882 + 0 + 22\*39\*50 = 67782

Третья диагональ

М(1,4)

К=1: M(1,1) + M(2,4) + p0p1p4 = 0 + 11616 + 8\*11\*29 = 14168

К = 2: M(1,2) + M(3,4) + p0p2p4 = 1 672 + 12 122 + 8\*19\*29 = 18202

К = 3: M(1,3) + M(4,4) + p0p3p4 = 5016 + 0 +8\*22\*29 = 10120

М(2,5)

К = 2: M(2,2) + M(3,5) + p1p2p5 = 0 + 33611 +11\*19\*39 = 41762

К = 3: M(2,3) + M(4,5) + p1p3p5 = 4 598 + 24 882 + 11\*22\*39 = 38918

К = 4: M(2,4) + M(5,5) + p1p4p5 = 11616 + 0 + 11\*29\*39 = 24057

М(3,6)

К = 3: M(3,3) + M(4,6) + p2p3p6 = 0 + 67782 +19\*22\*50 = 88682

К = 4: M(3,4) + M(5,6) + p2p4p6 = 12 122 + 56 550 + 19\*29\*50 = 96222

К = 5: M(3,5) + M(6,6) + p2p5p6 = 33611 + 0 +19\*39\*50 = 70661

Четвертая диагональ

М(1,5)

К = 1: M(1,1) + M(2,5) + p0p1p5 = 0 + 24057 +8\*11\*39 = 27489

К = 2: M(1,2) + M(3,5) + p0p2p5 = 1 672 + 33611 + 8\*19\*39 = 41211

К = 3: M(1,3) + M(4,5) + p0p3p5 = 5016 + 24 882 + 8\*22\*39 = 36762

К = 4: M(1,4) + M(5,5) + p0p4p5 = 10120 + 0 + 8\*29\*39 = 19168

М(2,6)

К = 2: M(2,2) + M(3,6) + p1p2p6 = 0 + 70661 + 11\*19\*50 = 81111

К = 3: M(2,3) + M(4,6) + p1p3p6 = 4 598 + 67782 + 11\*22\*50 = 84480

К = 4: M(2,4) + M(5,6) + p1p4p6 = 11616 + 56 550 + 11\*29\*50 = 84116

К = 5: M(2,5) + M(6,6) + p1p5p6 = 24057 + 0 +11\*39\*50 = 45507

М(1,6)

К = 1: M(1,1) + M(2,6) + p0p1p6 = 0 + 45507 + 8\*11\*50 = 49907

К = 2: M(1,2) + M(3,6) + p0p2p6 = 1 672 + 70661 + 8\*19\*50 = 79933

К = 3: M(1,3) + M(4,6) + p0p3p6 = 5016 + 67782 + 8\*22\*50 = 81598

К = 4: M(1,4) + M(5,6) + p0p4p6 = 10120 + 56 550 + 8\*29\*50 = 78270

К = 5: M(1,5) + M(6,6) + p0p5p6 = 19168 + 0 + 8\*39\*50 = 34768

Таблица К

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| i/j | 1 | 2 | 3 | 4 | 5 | 6 |
| 1 | 0 | 1 | 2 | 3 | 4 | 5 |
| 2 | x | 0 | 3 | 3 | 4 | 5 |
| 3 | x | X | 0 | 4 | 4 | 5 |
| 4 | X | x | x | 0 | 4 | 5 |
| 5 | x | x | x | X | 0 | 5 |
| 6 | x | x | x | x | x | 0 |