Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(ФГБОУ ВО «СибГУТИ»)

*Факультет информатики и вычислительной техники*

*Кафедра прикладной математики и кибернетики (ПМ и К)*

Расчетно-графическое задание по дисциплине

«Теория сложности вычислительных процессов и структур»

Выполнил:

студент гр. ИП-513

Санин И. В.

Проверил:

старший преподаватель

Разинкина Т. Э.

Новосибирск

2017

Оглавление

[I. Задача: 3](#_Toc497936753)

[II. Теоретическая часть. 3](#_Toc497936754)

[III. Листинг 4](#_Toc497936755)

[IV. Результаты 8](#_Toc497936756)

# Задача:

Написать 2 программы, вычисляющие произведение двух матриц A[i,j] = (-1)\*(i+j), B[i,j]=i-j, I,j=1..100. Использовать алгоритмы быстрого и обычного умножения. Сравнить трудоемкость двух алгоритмов.

# Теоретическая часть.

Умножение матриц — одна из основных операций над матрицами. Матрица, получаемая в результате операции умножения, называется произведением матриц.

Произведением матриц A и B размерностями и называется матрица С размерности , каждый элемент в которой равен .

Трудоемкость обычного умножения имеет порядок , т.к. в матрице-результате перемножения будет элементов и каждый из них вычисляется за операций попарного умножения и сложения.

Алгоритм Штрассена предназначен для быстрого умножения матриц. В основе алгоритма лежит рекурсивное разбиение матриц на блоки 2Х2. Штрассен доказал, что матрицы 2Х2 можно некоммутативно перемножить с помощью семи умножений, поэтому на каждом этапе рекурсии выполняется семь умножений вместо восьми. В результате, в отличие от традиционного алгоритма умножения матриц (по формуле работающего за время ), алгоритм Штрассена умножает матрицы за время , что даёт выигрыш на больших плотных матрицах начиная, примерно, от 64×64.

Формулы Штрассена:

Элементы, необходимые для вычисления блоков матрицы – произведения входных матриц:

*M1 = (A2 – A4)(B3 + B4)*

*M2 = (A1 + A4)(B1 + B4)*

*M3 = (A1 – A3)(B1 + B2)*

*M4 = (A1 + A2)B4*

*M5 = A1(B2 – B4)*

*M6 = A4 (B3 – B1)*

*M7 = (A3 + A4)B1*

Вычисление блоков конечной матрицы:

*C1 = M1 +M2 – M4 + M6*

*C2 = M4 + M5*

*C3 = M6 +M7*

*C4 = M2 – M3 + M5 – M7*

# Листинг

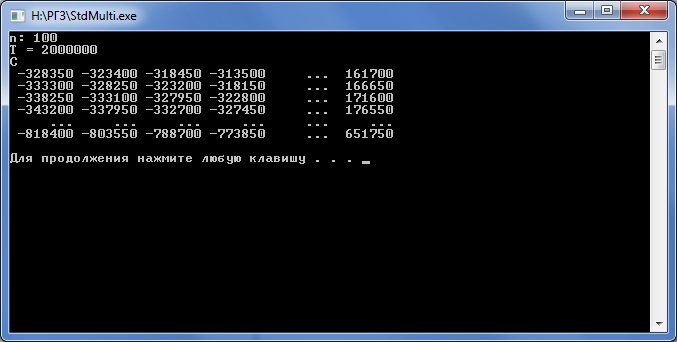
Программа, реализующая обычное умножение:

|  |
| --- |
| #include <iostream>  using namespace std;  int T = 0;  int \*\*SMM(int \*\*A, int \*\*B, int n);  int main(int argc, char const \*argv[])  {  int n = 100;  cout << "n: ";  cin >> n;  int i, j;  int \*\*A = new int \*[n];  int \*\*B = new int \*[n];  for(i = 0; i < n; ++i) {  A[i] = new int [n];  B[i] = new int [n];  for(j = 0; j < n; ++j) {  A[i][j] = -1 \* (i + j);  B[i][j] = i - j;  }  }  int \*\*C = SMM(A, B, n);  cout << "T = " << T << endl;  cout << "C" << endl;  if(n < 16) {  for(i = 0; i < n; ++i) {  for(j = 0; j < n; ++j) {  cout.width(5);  cout << C[i][j];  }  cout << endl;  }  } else {  for(i = 0; i < 4; ++i) {  for(j = 0; j < 4; ++j) {  cout.width(8);  cout << C[i][j];  }  cout << " ...";  cout.width(8);  cout << C[i][n-1];  cout << endl;  }  for(i = 0; i < 6; ++i) {  cout << " ...";  }  cout << endl;  for(j = 0; j < 4; ++j) {  cout.width(8);  cout << C[n-1][j];  }  cout << " ...";  cout.width(8);  cout << C[n-1][n-1];  cout << endl;  }  cout << endl;  system("PAUSE");  return 0;  }  int \*\*SMM(int \*\*A, int \*\*B, int n) {  int i, j, k;  int \*\*C = new int \*[n];  for(i = 0; i < n; ++i) {  C[i] = new int [n];  for(j = 0; j < n; ++j) {  C[i][j] = 0;  }  }  for(i = 0; i < n; ++i) {  for(j = 0; j < n; ++j) {  for(k = 0; k < n; ++k) {  C[i][j] += A[i][k] \* B[k][j];  T += 2;  }  }  }  return C;  } |

Программа, реализующая алгоритм Штрассена:

|  |
| --- |
| #include <iostream>  using namespace std;  int T = 0;  int \*\*FMM(int \*\*A, int \*\*B, int n);  int \*\*addsub(int \*\*A, int \*\*B, int n, int k);  int main(int argc, char const \*argv[])  {  int n = 100;  cout << "n: ";  cin >> n;  int bufn = 1, resn = n;  int raz = 0;  while(bufn < n) {  bufn <<= 1;  }  int i, j;  int \*\*A = new int \*[bufn];  int \*\*B = new int \*[bufn];  for(i = 0; i < n; ++i) {  A[i] = new int [bufn];  B[i] = new int [bufn];  for(j = 0; j < n; ++j) {  A[i][j] = -1 \* (i + j);  B[i][j] = i - j;  }  for(j = n; j < bufn; ++j) {  A[i][j] = 0;  B[i][j] = 0;  }  }  for(i = n; i < bufn; ++i) {  A[i] = new int [bufn];  B[i] = new int [bufn];  for(j = 0; j < bufn; ++j) {  A[i][j] = 0;  B[i][j] = 0;  }  }  n = bufn;  int \*\*C = FMM(A, B, n);  n = resn;  cout << "T = " << T << endl;  cout << "C" << endl;  if(n < 16) {  for(i = 0; i < n; ++i) {  for(j = 0; j < n; ++j) {  cout.width(5);  cout << C[i][j];  }  cout << endl;  }  } else {  for(i = 0; i < 4; ++i) {  for(j = 0; j < 4; ++j) {  cout.width(8);  cout << C[i][j];  }  cout << " ...";  cout.width(8);  cout << C[i][n-1];  cout << endl;  }  for(i = 0; i < 6; ++i) {  cout << " ...";  }  cout << endl;  for(j = 0; j < 4; ++j) {  cout.width(8);  cout << C[n-1][j];  }  cout << " ...";  cout.width(8);  cout << C[n-1][n-1];  cout << endl;  }  cout << endl;  system("PAUSE");  return 0;  }  int \*\*FMM(int \*\*A, int \*\*B, int n) {  int i, j, k;  int hn = n/2;  // result  int \*\*C = new int \*[n];  for(i = 0; i < n; ++i)  C[i] = new int [n];  if(n == 1) {  C[0][0] = A[0][0] \* B[0][0];  T++;  return C;  }  // Ak, Bk, Ck are 12 matrixes for fast multiplying  int \*\*\*Ak = new int \*\*[4];  int \*\*\*Bk = new int \*\*[4];  int \*\*\*Ck = new int \*\*[4];  // rA, rB are pointers to RIGHT half of matrixes A and B (for Ak, and Bk)  int \*\*rA = new int \*[n];  int \*\*rB = new int \*[n];    for(i = 0; i < n; ++i) {  rA[i] = A[i] + hn;  rB[i] = B[i] + hn;  }  for(i = 0; i < 4; ++i) {  if(!(i % 2)) {  Ak[i] = A;  Bk[i] = B;  } else {  Ak[i] = rA;  Bk[i] = rB;  }  if(i >= 2) {  Ak[i] += hn;  Bk[i] += hn;  }  }  int \*\*\*M = new int \*\*[7];  M[0] = FMM(addsub(Ak[1], Ak[3], hn, -1), addsub(Bk[2], Bk[3], hn, 1), hn);  M[1] = FMM(addsub(Ak[0], Ak[3], hn, 1), addsub(Bk[0], Bk[3], hn, 1), hn);  M[2] = FMM(addsub(Ak[0], Ak[2], hn, -1), addsub(Bk[0], Bk[1], hn, 1), hn);  M[3] = FMM(addsub(Ak[0], Ak[1], hn, 1), Bk[3], hn);  M[4] = FMM(Ak[0], addsub(Bk[1], Bk[3], hn, -1), hn);  M[5] = FMM(Ak[3], addsub(Bk[2], Bk[0], hn, -1), hn);  M[6] = FMM(addsub(Ak[2], Ak[3], hn, 1), Bk[0], hn);  Ck[0] = addsub( addsub( addsub(M[0], M[1], hn, 1), M[3], hn, -1), M[5], hn, 1);  Ck[1] = addsub(M[3], M[4], hn, 1);  Ck[2] = addsub(M[5], M[6], hn, 1);  Ck[3] = addsub( addsub( addsub(M[1], M[2], hn, -1), M[4], hn, 1), M[6], hn, -1);  for(i = 0; i < hn; ++i) {  for(j = 0; j < hn; ++j) {  C[i][j] = Ck[0][i][j];  C[i][j + hn] = Ck[1][i][j];  C[i + hn][j] = Ck[2][i][j];  C[i + hn][j + hn] = Ck[3][i][j];  }  }  return C;  }  int \*\*addsub(int \*\*A, int \*\*B, int n, int k) {  int \*\*C = new int \*[n];  for(int i = 0; i < n; ++i) {  C[i] = new int [n];  for(int j = 0; j < n; ++j) {  C[i][j] = A[i][j] + k \* B[i][j];  T++;  }  }  return C;  } |

# Результаты

Вывод программы – реализации стандартного умножения матриц: 

Вывод программы – реализации умножения матриц алгоритмом Штрассена:

