МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра математического обеспечения и применения ЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Построение и анализ алгоритмов»

Тема: Поиск с возвратом

Студент гр. 8304 _____ Алтухов А.Д.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Построение и анализ алгоритма, основанного на бэктрекинге, для решения задачи квадрирования квадрата минимально возможным количеством квадратов.

Вариант 3и.

Итеративный бэктрекинг. Исследование кол-ва операций от размера квадрата.

Основные теоретические положения.

Размер столешницы — одно целое число N (2 <= N <= 20). Необходимо найти и вывести: одно число K, задающее минимальное количество обрезков(квадратов), из которых можно построить столешницу(квадрат) заданног о размера *N*. Далее должны идти *K* строк, каждая из которых должна содержать три целых числа *x, y* и *w*, задающие координаты левого верхнего угла и длину стороны соответствующего обрезка(квадрата).

Описание алгоритма.

Был написан алгоритм, использующий бэктрекинг. Так как имеет смысл искать нужное разбиение только для квадратов со стороной, длина которой простое число, то рассматривать работу необходимо именно для такого случая. Алгоритм выполняется следующим образом:

1. Неотъемлемая часть каждого разбиения — три квадрата, расположенные в углах основного квадрата. Их стороны — N/2+1, N/2, N/2, больший квадрат располагается в левом верхнем углу, оставшиеся размещаются в соседние углы. Таким образом, обрабатываемая область сокращается в четыре раза.
2. В ближайшую свободную ячейку вставляется максимально возможный квадрат. Повторяется до полного заполнения квадрата.
3. Если количество квадратов, использованных в текущем заполнении, меньше, чем количество квадратов, установленное при предыдущих

заполнениях, то текущее поле копируется. При старте программы минимальное количество квадратов равняется значению 2*N — именно столько квадратов нужно для заполнения, если использовать квадрат со стороной N-1.

4. Бэктрекингом удаляются все единичные квадраты, первый неединичный квадрат заменяется квадратом со стороной на один меньше предыдущей.
5. Действия 2-4 повторяются, пока функция бэктрекинга не удалит один из квадратов, вставленных в поле на шаге 1. При этом активируется специальный флаг, являющийся условием выхода из алгоритма и гарантией того, что все варианты, не затрагивающие основные квадраты, уже перебраны.

Описание основных структур данных и функций.

struct Square — структура с информацией о заполняющем квадрате.

class Field — класс, представляющий собой квадрат, который необходимо заполнить. Включает в себя двумерный массив размером NxN для представления расположения квадратов и вектор, хранящий последовательность использованных при заполнении квадратов. В классе присутствуют следующие методы:

bool fill(int minAmount) — функция, осуществляющая выполнение шага 2 алгоритма. Возвращает информацию о том, было ли заполнено поле до конца, или количество вмещенных квадратов уже превысило минимально допустимое число.

void stepBack() — функция, выполняющая шаг 4 алгоритма.

Тестирование.

Таблица 1 – Результаты тестирование

Ввод	**Вывод**
5	8

	1 1 3 4 1 2 1 4 2 4 4 2 3 4 1 3 5 1 4 3 1 5 3 1
7	9 6 6 2 4 6 2 7 5 1 4 5 1 7 4 1 5 4 2 1 5 3 5 1 3 1 1 4 runtime = 0.005 operations: 88
11	11 9 9 3 6 9 3 11 8 1 10 8 1 6 8 1 6 7 1 10 6 2

	7 6 3 1 7 5 7 1 5 1 1 6 runtime = 0.014 operations: 1551
12	4 1 1 6 7 1 6 1 7 6 7 7 6
23	13 19 19 5 19 17 2 12 17 7 21 16 3 20 16 1 12 14 3 12 13 1 20 12 4 15 12 5 13 12 2 1 13 11 13 1 11 1 1 12 runtime = 0.425 operations: 435225

Исследование.

Было проведено исследование зависимости количества операций от длины стороны квадрата. За операцию принято считать попытку вставить заполняющий квадрат. На рис. 1 приведен график зависимости количества операций от стороны квадрата.

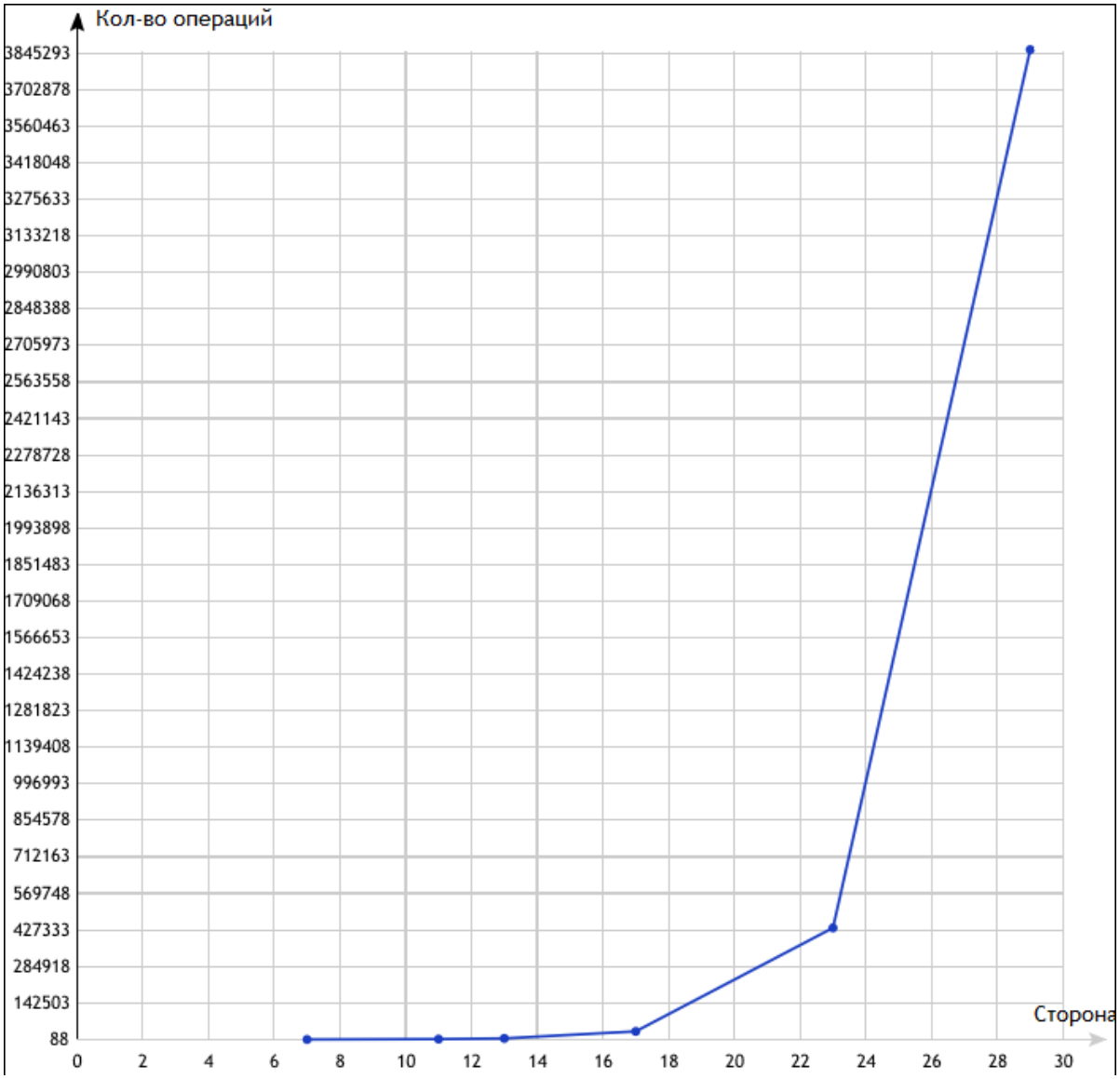

Рисунок 1 — График зависимости количества операций от стороны квадрата.

Из графика видно, что сложность алгоритма носит экспоненциальный характер. Затраты памяти, из приведенного выше, $O(N^2)$. Исходный код исследуемого алгоритма приведен в приложении А.

Вывод.

В ходе работы был построен и исследован алгоритм квадрирования квадрата с использованием бэктрекинга.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД

```
#include <iostream>
#include <algorithm>
#include <cmath>
#include <vector>
#include <locale>
#include <ctime>

struct Square {
	int x;
	int y;
	int w;
	bool mainSquare;
};

class Field {

	std::vector< std::vector<int> > arr;

	int width;
	int lastX;
	int lastY;
	int startX;
	int startY;
	int squaresAmount;
	int currentPiece;

	std::vector< Square > consequence;

	bool itsTimeToStop;

	unsigned long int operationsNum;

public:

	Field(int n) {

		width = n;
		arr.resize(n);

		for (int i = 0; i < n; i++) {
			arr[i].resize(n);
		}

		for (int i = 0; i < width; i++) {

			for (int j = 0; j < width; j++) {
```

```
                arr[i][j] = 0;
            }
        }

        lastX = 0;
        lastY = 0;
        squaresAmount = 0;
        currentPiece = 0;
        itsTimeToStop = false;

        operationsNum = 0;
}

void setStartPoint(int val) {
        startX = val;
        startY = val;
}

bool isEmpty() {
        return consequence.empty();
}
int getOperationsNum() {
        return operationsNum;
}
int getSquaresAmount() {
        return squaresAmount;
}
int getCurrentPiece() {
        return currentPiece;
}
void setLastX(int val) {
        lastX = val;
}
void setLastY(int val) {
        lastY = val;
}
void setCurrentPiece(int val) {
        currentPiece = val;
}
bool isItTimeToStop() {
        return itsTimeToStop;
}

bool fill(int minAmount) {

        for (int i = startY; i < width; i++) {

            for (int j = startX; j < width; j++) {
                if (!arr[i][j]) {
                    lastY = i;
                    lastX = j;
```

```
                        if (squaresAmount >= minAmount) {
                              return false;
                        }

                        while (!arr[i][j]) {
                              operationsNum++;
                              if (tryToFit()) {
                                    setSquare();
                                    currentPiece = width / 2;
                                    break;
                              }
                              else {
                                    currentPiece--;
                              }
                        }
                  }
            }
      }
      return true;

}

bool tryToFit() {

      if (lastX + currentPiece > width) { return false; }
      if (lastY + currentPiece > width) { return false; }

      for (int i = lastY; i < lastY + currentPiece; i++) {
            for (int j = lastX; j < lastX + currentPiece; j++) {
                  if (arr[i][j]){
                        return false;
                  }
            }
      }
      return true;

}

void setSquare(bool mainSquare = false) {

      squaresAmount++;
      for (int i = lastY; i < lastY + currentPiece; i++) {
            for (int j = lastX; j < lastX + currentPiece; j++) {
                  arr[i][j] = squaresAmount;
            }
      }

      consequence.push_back({ lastX, lastY, currentPiece, mainSquare
});

}
```

```
void print() {
	for (int i = 0; i < width; i++) {

		for (int j = 0; j < width; j++) {
			std::cout << " " << arr[i][j];
		}
		std::cout << "\n";
	}
}

void resetSquare(int x, int y, int w) {

	squaresAmount--;

	for (int i = y; i < y + w; i++) {
		for (int j = x; j < x + w; j++) {
			arr[i][j] = 0;
		}
	}

	consequence.pop_back();

}

void stepBack() {

	if (isEmpty()) { return; }
	Square lastSquare = consequence.back();
	if (lastSquare.mainSquare) {
		itsTimeToStop = true;
	}
	resetSquare(lastSquare.x, lastSquare.y, lastSquare.w);

	lastX = lastSquare.x;
	lastY = lastSquare.y;


	if (lastSquare.w == 1) {
		stepBack();
	}
	else{
		currentPiece = lastSquare.w - 1;
		setSquare();
	}

}

void printAnswer() {
	while (!(isEmpty())) {
		Square sq = consequence.back();
```

```
                consequence.pop_back();
                std::cout << sq.x + 1 << " " << sq.y + 1 << " " << sq.w;
                std::cout << "\n";
            }
        }
};

int main() {

    setlocale(LC_ALL, "Russian");

    int n;
    //std::cout << "Введите размер столешницы\n";
    std::cin >> n;

    if (n <= 1) {
        std::cout << "Некорректный размер квадрата\n";
        return 0;
    }

    int time = clock();

    Field field(n);

    Field minField(n);
    if (n % 2 == 0) {
        std::cout << 4 << "\n";
        std::cout << 1 << " " << 1 << " " << n / 2 << "\n";
        std::cout << 1 + n / 2 << " " << 1 << " " << n / 2 << "\n";
        std::cout << 1 << " " << 1 + n / 2 << " " << n / 2 << "\n";
        std::cout << 1 + n / 2 << " " << 1 + n / 2 << " " << n / 2 <<
"\n";
        return 0;
    }
    else if (n % 3 == 0) {
        std::cout << 6 << "\n";
        std::cout << 1 << " " << 1 << " " << 2 * n / 3 << "\n";
        std::cout << 1 + 2 * n / 3 << " " << 1 << " " << n / 3 << "\n";
        std::cout << 1 << " " << 1 + 2 * n / 3 << " " << n / 3 << "\n";
        std::cout << 1 + 2 * n / 3 << " " << 1 + n / 3 << " " << n / 3
<< "\n";
        std::cout << 1 + n / 3 << " " << 1 + 2 * n / 3 << " " << n / 3
<< "\n";
        std::cout << 1 + 2 * n / 3 << " " << 1 + 2 * n / 3 << " " << n /
3 << "\n";
        return 0;

    }
    else if (n % 5 == 0) {
        std::cout << 8 << "\n";
        std::cout << 1 << " " << 1 << " " << 3 * n / 5 << "\n";
```

```
            std::cout << 1 + 3 * n / 5 << " " << 1 << " " << 2 * n / 5 <<
"\n";
            std::cout << 1 << " " << 1 + 3 * n / 5 << " " << 2 * n / 5 <<
"\n";
            std::cout << 1 + 3 * n / 5 << " " << 1 + 3 * n / 5 << " " << 2 *
n / 5 << "\n";
            std::cout << 1 + 2 * n / 5 << " " << 1 + 3 * n / 5 << " " << n /
5 << "\n";
            std::cout << 1 + 2 * n / 5 << " " << 1 + 4 * n / 5 << " " << n /
5 << "\n";
            std::cout << 1 + 3 * n / 5 << " " << 1 + 2 * n / 5 << " " << n /
5 << "\n";
            std::cout << 1 + 4 * n / 5 << " " << 1 + 2 * n / 5 << " " << n /
5 << "\n";
            return 0;
      }


      field.setLastX(0);
      field.setLastY(0);
      field.setCurrentPiece(std::ceil(static_cast<double>(n) / 2));
      field.setSquare(true);

      field.setStartPoint(n / 2);

      field.setCurrentPiece((n / 2));
      field.setLastX(std::ceil(static_cast<double>(n) / 2));
      field.setSquare(true);

      field.setLastX(0);
      field.setLastY(std::ceil(static_cast<double>(n) / 2));
      field.setSquare(true);

      //field.print();
      //std::cout << "\n=======================================\n";


      field.setLastX(0);
      field.setLastY(0);

      int minAmount = 2*n;

      field.setCurrentPiece(n/2);


      do{
            field.setCurrentPiece(n / 2);

            bool fillToFull = field.fill(minAmount);

            if (fillToFull) {
```

```
                if (field.getSquaresAmount() < minAmount) {
                      minField = field;
                      //std::cout << "\nЛУЧШЕЕ\n";
                      //minField.print();
                      //std::cout <<
"\n=======================================\n";
                      minAmount = field.getSquaresAmount();
                }
          }


          field.stepBack();

          //field.print();
          //std::cout << "\n=======================================\n";


     } while ((!field.isItTimeToStop())  );




     //std::cout << "Минимальное количество квадратов: " << minAmount <<
"\n";
     //std::cout << "Расстановка: \n";
     //minField.print();

     std::cout << minAmount << "\n";
     minField.printAnswer();
     std::cout << std::endl << "runtime = " << (clock()-time) / 1000.0 <<
std::endl; // время работы программы
     std::cout << "operations: " << field.getOperationsNum() << "\n";
     return 0;
}
```