МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №1

по дисциплине «Построение и анализ алгоритмов»

ТЕМА: ПЕРЕБОР С ВОЗВРАТОМ

Вариант 2и

Студент гр. 8304 Рыжиков А. В.
Преподаватель Фирсов К. В.

Санкт-Петербург

2019

1 Цель работы.

Решить задачу на перебор с возвратом. Получить опыт построения алгоритмов и их оптимизации.

2 Задача

Квадрирование квадрата. Необходимо разбить квадрат со стороной длины N на квадраты минимальным образом, и предоставить это разбиение.

3 Алгоритм разбиения.

Рассматривается сторона квадрата. Если сторона квадрата число чётное, то квадрат гарантировано разбивается на 4 квадрата. Если сторона квадрата число нечётное, то смотрим на то, является ли оно простым или нет . Если оно непростое и делится на 3, то квадрат гарантировано делится на 6 квадратом, ели не делится на 3 то на 8. В обоих случаях используются квадраты кратные наибольшему делителю числа. Если сторона квадрата число простое, то запускается алгоритм перебора с возвратом.

Алгоритм

Шаг 1: В верхний левый угол ставится квадрат со стороной N/2 + 1. Также устанавливаются квадраты со стороной N/2 в верхний правы и нижний левый угол. Остаётся область в нижнем правом углу (квадрат с верезанным уголком), в которой будет происходить перебор.

Шаг 2: Длина свободно стороны в верхней части составляет N-N/2-1. Генерируются все возможные комбинации чисел дающие данное число. В данную область устанавливаются квадраты из комбинации.

Шаг 3: Устанавливается максимально возможный квадрат в нижний правый угол.

Шаг 4: Происходит забивание квадрата квадратами за линейное время.

Шаг 5: Подсчитывается количество квадратов в разбиении, если оно меньше минимального, то это число min = количество квадратов в разбиении.

Шаг 6: Перебираются все комбинаци

Временная сложность

[Количество всех комбинаций дающие число N/2] * 2 * O(1)

Вывод: *в ходе работы был получен опыт работы по построению алгоритмов и их оптимизации.*

Приложение

Код программы lab1.cpp

```cpp
#include <iostream>
#include <cmath>
#include <vector>

using namespace std;

std::vector<std::vector<int>> recursia2(int number);

bool isPrimeNumber(int number) {
    int i;
    bool isPrime = true;
    for (i = 2; i <= (sqrt(abs(number))); i++) {
        if (number % i == 0) {
            isPrime = false;
            break;
        }
    }
    return isPrime;
}

int findLargestDivisor(int number) {
    int i;
    int largestDivisor = 0;
    for (i = 3; i * i <= number; ++i)
        if (number % i == 0) break;
    if (i * i <= number) {
        largestDivisor = number / i;
    }

    return largestDivisor;
}

void evenSize(int sizeSquare) {
    //std::cout << "n=" << sizeSquare << "\n";
    std::cout << 4 << "\n";
    int sizeMin = sizeSquare / 2;


    std::cout << 0 << " " << 0 << " " << sizeMin << "\n";
    std::cout << 0 << " " << sizeMin << " " << sizeMin << "\n";
    std::cout << sizeMin << " " << 0 << " " << sizeMin << "\n";
    std::cout << sizeMin << " " << sizeMin << " " << sizeMin << "\n";
```

```
}

void notEvenAndNotPrime(int sizeSquare) {
    //std::cout << "n=" << sizeSquare << "\n";
    int largestDivisor = findLargestDivisor(sizeSquare);


    if (sizeSquare % 3 == 0) {
        int bigSize = sizeSquare - largestDivisor;
        std::cout << 6 << "\n";
        std::cout << 0 << " " << 0 << " " << bigSize << "\n";
        std::cout << bigSize << " " << 0 << " " << largestDivisor << "\n";
        std::cout << 0 << " " << bigSize << " " << largestDivisor << "\n";
        std::cout << bigSize << " " << bigSize - largestDivisor << " " << largestDivi-
sor << "\n";
        std::cout << bigSize - largestDivisor << " " << bigSize << " " << largestDivi-
sor << "\n";
        std::cout << bigSize << " " << bigSize << " " << largestDivisor << "\n";
    } else {
        int bigSize = sizeSquare - largestDivisor * 2;
        std::cout << 8 << "\n";
        std::cout << 0 << " " << 0 << " " << bigSize << "\n";
        std::cout << 0 << " " << bigSize << " " << largestDivisor * 2 << "\n";
        std::cout << bigSize << " " << 0 << " " << largestDivisor * 2 << "\n";
        std::cout << bigSize << " " << bigSize << " " << largestDivisor * 2 << "\n";

        std::cout << bigSize << " " << bigSize - largestDivisor << " " << largestDivi-
sor << "\n";
        std::cout << bigSize + largestDivisor << " " << bigSize - largestDivisor << " "
<< largestDivisor
                  << "\n";
        std::cout << bigSize - largestDivisor << " " << bigSize << " " << largestDivi-
sor << "\n";
        std::cout << bigSize - largestDivisor << " " << bigSize + largestDivisor << " "
<< largestDivisor
                  << "\n";


    }

}

void setNumbers(int **matrix, int x, int y, int size, int numbers) {
    for (int i = y; i < y + size; ++i) {
        for (int j = x; j < x + size; ++j) {
            matrix[j][i] = numbers;
        }
    }
}

void setSquare(int **matrix, int x, int y, int size) {
    setNumbers(matrix, x, y, size, size);
}

void draw(int **matrix, int sizeSquare) {
    for (int i = 0; i < sizeSquare; ++i) {
        for (int j = 0; j < sizeSquare; ++j) {
            std::cout << matrix[j][i] << "  ";
        }
        std::cout << "\n";
    }
```

```
}

void resetMatix(int **matrix, int sizeSquare) {
    int bigSize = sizeSquare / 2 + 1;
    int bigSize2 = sizeSquare - bigSize;

    setNumbers(matrix, bigSize, bigSize, sizeSquare - bigSize, 0);
    for (int i = bigSize2 + 1; i < sizeSquare; ++i) {
        matrix[bigSize2][i] = 0;
        matrix[i][bigSize2] = 0;
    }


}

void prepareAnswers(int **matrix, int sizeSquare) {
    for (int i = 0; i < sizeSquare; ++i) {
        for (int j = 0; j < sizeSquare; ++j) {
            if (matrix[j][i] != 0) {
                std::cout << j << " " << i << " " << matrix[j][i] << "\n";
                setNumbers(matrix, j, i, matrix[j][i], 0);
            }
        }
    }
}

bool isEmptyCells(int **matrix, int x1, int y1, int size) {
    for (int i = y1; i < y1 + size; ++i) {
        for (int j = x1; j < x1 + size; ++j) {
            if (matrix[j][i] != 0) {
                return false;
            }
        }
    }
    return true;
}

void copyMatrix(int **matrix, int **matrix2, int sizeSquare) {
    for (int i = 0; i < sizeSquare; ++i) {
        for (int j = 0; j < sizeSquare; ++j) {
            matrix2[j][i] = matrix[j][i];
        }
    }
}

void setLocalSquares3(int **matrix, int x, int y, int sizeSquare) {
    int count = 0;
    for (int i = 0; i < sizeSquare; ++i) {
        if (x + i < sizeSquare && y - i > 0) {
            if (isEmptyCells(matrix, x, y - i, i + 1)) {
                count++;
            } else {
                break;
            }
        } else {
            break;
        }
    }

    count--;
```

```
    setSquare(matrix, x, y - count, count + 1);
}

void setLocalSquares5(int **matrix, int x, int y, int sizeSquare) {
    int count = 0;
    for (int i = 0; i < sizeSquare; ++i) {
        if (x + i < sizeSquare && y + i < sizeSquare) {
            if (isEmptyCells(matrix, x, y, i + 1)) {
                count++;
            } else {
                break;
            }
        } else {
            break;
        }
    }

    count--;

    setSquare(matrix, x, y, count + 1);

}


void
completeSquares3(int **matrix, int **matrix2, int sizeSquare, int *MinCount, int count,
const int *x, const int *y) {
    for (int i = sizeSquare - 1; i >= *y; --i) {
        bool isEnd = false;
        for (int j = *x - 1; j < sizeSquare; ++j) {
            if (matrix[j][i] == 0) {
                //draw(matrix,sizeSquare);
                setLocalSquares3(matrix, j, i, sizeSquare);
                count++;
                if (count >= *MinCount) {
                    isEnd = true;
                    break;
                }
            }
        }
        if (isEnd) {
            break;
        }
    }
    if (count < *MinCount) {
        *MinCount = count;
        copyMatrix(matrix, matrix2, sizeSquare);
    }
}


void
completeSquares5(int **matrix, int **matrix2, int sizeSquare, int *MinCount, int count,
const int *x, const int *y) {
    for (int i = *y; i < sizeSquare; ++i) {
        bool isEnd = false;
        for (int j = *x - 1; j < sizeSquare; ++j) {
            if (matrix[j][i] == 0) {
                setLocalSquares5(matrix, j, i, sizeSquare);
                count++;
                if (count >= *MinCount) {
```

```
                    isEnd = true;
                    break;
                }
            }
        }
        if (isEnd) {
            break;
        }
    }
    if (count < *MinCount) {
        *MinCount = count;
        copyMatrix(matrix, matrix2, sizeSquare);
    }
}


int checkSuper(int **matrix, int **matrix2, int sizeSquare) {

    int MinCount = 1000;


    int x = sizeSquare / 2 + 1;
    int y = sizeSquare / 2;
    std::vector<std::vector<int>> combinations = recursia2(y);

    for (const std::vector<int> &numbers : combinations) {
        int localX = x;
        int localY = y;
        int count = 0;
        int max = 0;
        for (int i : numbers) {
            if (i == 0) {
                continue;
            }
            setSquare(matrix, localX, localY, i);
            count++;
            localX = localX + i;
            if (i > max) {
                max = i;
            }
        }
        max = y - max + 1;
        setSquare(matrix, sizeSquare - max, sizeSquare - max, max);
        count++;
        completeSquares5(matrix, matrix2, sizeSquare, &MinCount, count, &x, &y);
        resetMatix(matrix, sizeSquare);

        localX = x;
        localY = y;
        count = 0;
        max = 0;
        for (int i : numbers) {
            if (i == 0) {
                continue;
            }
            setSquare(matrix, localX, localY, i);
            count++;
            localX = localX + i;
            if (i > max) {
                max = i;
            }
```

```
        }
        max = y - max + 1;
        setSquare(matrix, sizeSquare - max, sizeSquare - max, max);
        count++;
        completeSquares3(matrix, matrix2, sizeSquare, &MinCount, count, &x, &y);
        resetMatix(matrix, sizeSquare);

    }


    return MinCount;
}

void notEvenAndPrime(int sizeSquare) {
    //std::cout << "n=" << sizeSquare << "\n";

    int countSquares = 3;

    int **matrix = new int *[sizeSquare];
    for (int i = 0; i < sizeSquare; ++i) {
        matrix[i] = new int[sizeSquare];
    }
    int **matrix2 = new int *[sizeSquare];
    for (int i = 0; i < sizeSquare; ++i) {
        matrix2[i] = new int[sizeSquare];
    }
    setNumbers(matrix, 0, 0, sizeSquare, 0);
    setNumbers(matrix2, 0, 0, sizeSquare, 0);

    int bigSize = sizeSquare / 2 + 1;
    int bigSize2 = sizeSquare - bigSize;

    setSquare(matrix, 0, 0, bigSize);
    setSquare(matrix, bigSize2 + 1, 0, bigSize2);
    setSquare(matrix, 0, bigSize2 + 1, bigSize2);


    countSquares = countSquares + checkSuper(matrix, matrix2, sizeSquare);


    std::cout << countSquares << "\n";
    //draw(matrix2, sizeSquare);
    //std::cout << "\n";
    prepareAnswers(matrix2, sizeSquare);

    for (int k = 0; k < sizeSquare; ++k) {
        delete (matrix[k]);
        delete (matrix2[k]);

    }
    delete[](matrix);
    delete[](matrix2);

}

void mainCheck(int sizeSquare) {
    if (sizeSquare % 2 == 0) {
        evenSize(sizeSquare);
    } else {
        if (isPrimeNumber(sizeSquare)) {
            notEvenAndPrime(sizeSquare);
```

```
        } else {
            notEvenAndNotPrime(sizeSquare);
        }
    }
}


std::vector<pair<int, int >> getAllCombinations(int number) {
    std::vector<pair<int, int>> allCombination(0);
    for (int i = 0; i < number; ++i) {
        int number2 = number - i;
        allCombination.emplace_back(std::make_pair(i, number2));
    }
    return allCombination;
}

void recursia(int number) {
    std::vector<pair<int, int >> combinations = getAllCombinations(number);
    for (pair<int, int> pair: combinations) {
        if (pair.first != 0 && pair.first != 1) {
            recursia(pair.first);
            std::cout << ":" << pair.second << ",";
        } else {
            std::cout << pair.first << " " << pair.second << ",";
        }
    }
}

std::vector<std::vector<int>> recursia2(int number) {
    std::vector<pair<int, int >> combinationsNumbers = getAllCombinations(number);
    std::vector<std::vector<int>> combinations2(0);
    for (pair<int, int> pair5: combinationsNumbers) {
        if (pair5.first == 0 || pair5.first == 1) {
            std::vector<int> locale1(0);
            locale1.emplace_back(pair5.first);
            locale1.emplace_back(pair5.second);
            combinations2.emplace_back(locale1);
        } else {
            std::vector<std::vector<int>> combinations3 = recursia2(pair5.first);
            for (const std::vector<int> &vector : combinations3) {
                std::vector<int> locale1 = vector;
                locale1.emplace_back(pair5.second);
                combinations2.emplace_back(locale1);
            }

        }
    }

    return combinations2;
}

int main() {
    int number;
    std::cin >> number;

    mainCheck(number);
    /*for (int i = 2; i < 21; ++i) {
        std::cout << i << "\n";
        mainCheck(i);
        std::cout <<"\n_______________\n";
```

```
    }*/

    /*for (int i = 30; i <= 40; ++i) {
        std::cout << "n=" << i << "\n";
        mainCheck(i);
    }*/

    //notEvenAndNotPrime(9);


    return 0;
}
```