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 Нам Ё Себ

Преподаватель Фирсов М.А.

Санкт-Петербург

2020

Цель работы.

Построение и анализ алгоритма бэктрекинг (поиск с возвратом) на основе решения задачи квадратирования квадратов.

Вариант 3р.

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.

Для решения поставленной задачи был использован бэктрекинг – алгоритм, осуществляющий перебор всех возможных вариантов. Для данной задачи была реализована функция search, которая осуществляет следующую последовательность действий:

1) Находит первую свободную ячейку в матрице.
2) Пытается вставить на ее место квадрат со стороной 1, 2, …
3) Зарисовывает квадрат с ранее найденной стороной.
4) При возникновении пересечений и/или других ошибок выполнение передается функции вызывающей данную.
5) Записывает результат в массив min_result.

Описание основных структур данных и функций.

Рекурсивная функция search – принимающая одномерный массив, соответствующий текущей зарисовке квадрата, одномерный массив current_result, хранящий данные о квадратах уже поучаствовавших в текущей зарисовке, массив

min_result, хранящий в себе данные о текущей минимальной зарисовке всего квадрата, параметр n – длину стороны закрашиваемого квадрата и параметр step, отвечающий за шаг увеличения длины для рассматриваемого квадрата.

Тестирование.

Таблица 1 – Результаты тестирование

Ввод	**Вывод**
5	8 0 0 2 2 0 3 0 2 1 0 3 1 0 4 1 1 2 1 1 3 2 3 3 2 Iter: 213
11	11 0 0 5 5 0 6 0 5 1 0 6 1 0 7 1 0 8 3 1 5 3 3 8 3 4 5 1 4 6 2 6 6 5

	Iter: 22351
17	12 0 0 8 8 0 9 0 8 2 0 10 2 0 12 5 2 8 4 5 12 1 5 13 4 6 8 2 6 10 3 8 9 1 9 9 8 Iter: 885526
21	6 0 0 7 0 7 7 0 14 7 7 0 7 7 7 14 14 0 7 Iter: 156
25	8 0 0 5 0 5 5 0 10 5 0 15 10 5 0 10

	10 10 15 15 0 10 Iter: 3369

Вывод.

В ходе работы был построен и анализирован алгоритм бэктрекинг на основе решения задачи квадратирования квадратов. Исходный код программы представлен в приложении А.

ПРИЛОЖЕНИЕ А.
ИСХОДНЫЙ КОД

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

struct Square
{
	int x;
	int y;
	int lenght;
};

void search(std::vector<int>& tabletop, std::vector<Square>& current_result,
std::vector<Square>& min_result, int n, int step, std::vector<int>& iter)
{
	Square check_square;
	bool fl = false;

	iter[0]++;
	for (int i = 0; i < n; ++i)
	{
		for (int j = 0; j < n; ++j)
		{
			if (tabletop[i * n + j] == 0)
			{
				check_square.x = i;
				check_square.y = j;
				fl = true;
				break;
			}
		}

		if (fl)
			break;
	}

	if (fl == false)
	{
		min_result = current_result;
		return;
	}

	for (int current_lenght = step; current_lenght < n; current_lenght += step)
	{
		iter[0]++;
		if (check_square.x + current_lenght > n || check_square.y +
current_lenght > n)
			return;

		iter[0]++;
		if (current_result.size() + 1 >= min_result.size())
			return;

		iter[0]++;
		for (int i = check_square.x; i < check_square.x + current_lenght; ++i)
			for (int j = check_square.y; j < check_square.y + current_lenght;
++j)
				if (tabletop[i * n + j] == 1)
					return;
```

```
			iter[0]++;
			for (int i = check_square.x; i < check_square.x + current_lenght; ++i)
				for (int j = check_square.y; j < check_square.y + current_lenght;
++j)
					tabletop[i * n + j] = 1;


			check_square.lenght = current_lenght;
			current_result.push_back(check_square);

			search(tabletop, current_result, min_result, n, step, iter);

			iter[0]++;
			for (int i = check_square.x; i < check_square.x + current_lenght; ++i)
				for (int j = check_square.y; j < check_square.y + current_lenght;
++j)
					tabletop[i * n + j] = 0;

			current_result.pop_back();

		}
}


int check(int n)
{
		int a = 1;
		for (int i = 2; i < n; ++i)
		{
			if (n % i == 0)
				a = i;
		}

		return a;
}


int main()
{
		int n = 0;
		std::cin >> n;

		std::vector<int> iter(1);

		iter[0]++;
		int min_side = check(n);

		std::vector<Square> current_result;
		std::vector<Square> min_result(n * 2 + 1);
		std::vector<int> tabletop(n * n, 0);



		if (min_side == 1 && n != 2)
		{

			for (int i = 0; i < n / 2; ++i)
				for (int j = 0; j < n / 2; ++j)
					tabletop[i * n + j] = 1;

			for (int i = n / 2; i < n; ++i)
				for (int j = 0; j < n / 2 + 1; j++)
```

```
                        tabletop[i * n + j] = 1;

                current_result.push_back({ 0, 0, n / 2 });
                current_result.push_back({ n / 2, 0, n / 2 + 1 });
                iter[0] += 2;
        }

        search(tabletop, current_result, min_result, n, min_side, iter);

        std::cout << min_result.size() << "\n";
        for (auto i : min_result)
                std::cout << i.x << " " << i.y << " " << i.lenght << "\n";

        std::cout << iter[0];
        return 0;
}
```