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

ОТЧЕТ

по лабораторной работе №1

по дисциплине «ПиАА»

Тема: Бэктрекинг

Студент(ка) гр. 0000 _____ Ивченко А.А.

Преподаватель _____ Размочаева Н.В.

Санкт-Петербург
2020

Цель работы.

Ознакомиться с работой алгоритма поиска с возвратом, научится применять полученные знания в решении задач на перебор всех возможных вариантов.

Формулировка задания.

У Вовы много квадратных обрезков доски. Их стороны (размер) изменяются от 1 до $N-1$, и у него есть неограниченное число обрезков любого размера. Но ему очень хочется получить большую столешницу - квадрат размера N. Он может получить ее, собрав из уже имеющихся обрезков(квадратов).

Например, столешница размера 7×7 может быть построена из 9 обрезков.

Внутри столешницы не должно быть пустот, обрезки не должны выходить за пределы столешницы и не должны перекрываться. Кроме того, Вова хочет использовать минимально возможное число обрезков.

Вариант 2И. Итеративный бэктрекинг. Исследование времени выполнения от размера квадрата.

Описание алгоритма.

В ходе работы был реализован класс Matrix размера N*N.

Для решения поставленной задачи был разработан алгоритм, осуществляющий поиск с возвратом итеративным методом. Для упрощения и сокращения количества итераций примерно ¾ матрицы заполняется 3-мя квадратами. Для больших значений N в свободный угол ставится 4-ый квадрат.

Работа алгоритма заключается в последовательном заполнении свободных областей по возможности максимально большими квадратами и уменьшении сторон наименьших квадратов на 1 (в случае, если квадрат единичный, он удаляется) до тех пор, пока матрица не станет заполненной полностью. Наименьшее количество обрезков, то есть лучшее решение в процессе перебора вариантов расстановок устанавливается как максимальное значение для следующих проверок. Алгоритм прекращает свою работу, когда текущее число квадратов в стеке превосходит лучшее решение.

Алгоритм оптимизирован для не простых чисел. Оценочная сложность алгоритма $O(n^3)$ в лучшем случае.

Тестирование

Рисунок 1 — результат теста №1

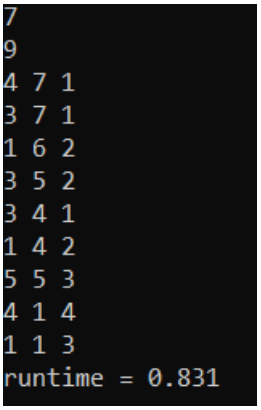


Рисунок 2 — результат теста №2

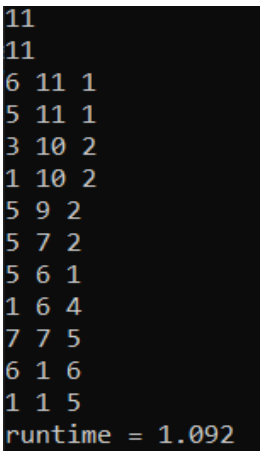
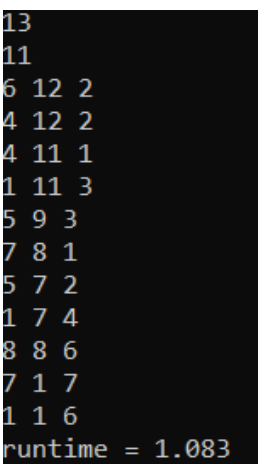


Рисунок 3 — результат теста №3



Вывод.

В ходе лабораторной работы был разобран алгоритм поиска с возвратом, в частности итеративный его метод реализации. Была составлена программа, выполняющая поиск наилучшей конфигурации квадратов в заданных границах, а также считающая время работы алгоритма.

Исходный код

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

using namespace std;

struct Square{

     int x,y,len;

};

class Matrix{

private:

     int n;
     int** matrix;

     int count,new_count;
     vector <Square> sq_arr;

     vector <vector<Square>> variants;

public:

     int choice;
```

```
          Matrix(int length, int count) :n(length),
new_count(length* length), count(0){

               matrix = new int* [length];

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

                    matrix[i] = new int[length];
                    for (int j = 0; j < length; j++){

                         matrix[i][j] = 0;
                    }
               }
          }

          void set_square(Square sq){

               count++;
               for (int i = sq.y; i < sq.y + sq.len; i++){

                    for (int j = sq.x; j < sq.x + sq.len; j++){

                         matrix[i][j] = count;

                    }
               }
          }

          void rem_square(Square sq){

               for (int i = sq.y; i < sq.y + sq.len; i++){

                    for (int j = sq.x; j < sq.x + sq.len; j++){
```

```
                matrix[i][j] = 0;
            }
        }
        count--;
    }

    bool check(int x, int y, int m) {

        if (x >= n || y >= n)
            return false;

        if (x + m > n || y + m > n)
            return false;

        for (int i = y; i < y + m; i++) {

            for (int j = x; j < x + m; j++) {

                if (matrix[i][j] != 0) {

                    return false;

                }
            }
        }

        return true;
    }


    bool is_filled() {

        for (int i = n - 1; i >= 0; --i)
```

```
            for (int j = n - 1; j >= 0; --j)

                if (matrix[i][j] == 0)
                    return false;

    return true;


}

void backtracking(std::ostream &out)
{

    Square sq;

    sq.len = ceil(n / 2);
    sq.x = 0;
    sq.y = 0;
    sq_arr.push_back(sq);
    set_square(sq);

    sq.x = n / 2;
    sq.y = 0;
    sq.len = n / 2 + 1;
    sq_arr.push_back(sq);
    set_square(sq);

    sq.x = n/2+1;
    sq.y = n / 2 + 1;
    sq.len = n / 2;
    sq_arr.push_back(sq);
    set_square(sq);
```

```
                if (n > 15) {
                     sq.len = n / 4+1;
                     sq.x = 0;
                     sq.y = 3*n / 4;
                     sq_arr.push_back(sq);
                     set_square(sq);
                     count = 4;
                }
```

```
                while ((count < new_count) && (!is_filled())) {
                          for (int y = 0; y < n; y++) {
                               for (int x = 0; x < n; x++) {
                                    if (matrix[y][x] == 0) {
                                         for (int size_square = n;
size_square > 0; size_square--) {
                                              if (check(x, y,
size_square)) {
                                                   Square sq{ x,y,
size_square };
                                                   set_square(sq);
     sq_arr.push_back(sq);
```

```
                                        break;
                                   }

                              }

                         }
                    }
               }

               if (choice != 0) {
                    for (int i = 0; i < n; i++) {

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

               if (count < new_count)
               {
                    new_count = count;
                    variants.push_back(sq_arr);
               }


               while (!sq_arr.empty() &&
sq_arr[sq_arr.size() - 1].len < n/5 + 1) {

                    rem_square(sq_arr[sq_arr.size() -
1]);
                    sq_arr.pop_back();
```

```
                    }

                    if (!(sq_arr.empty() &&
(sq_arr[sq_arr.size() - 1]).len < n/2) ){//уменьшение стороны
верхнего квадрата в стеке на 1

                         sq = sq_arr[sq_arr.size() - 1];
                         sq_arr.pop_back();
                         rem_square(sq);
                         sq.len -= 1;
                         set_square(sq);
                         sq_arr.push_back(sq);
                    }

               }
          }

          void print(int k){

               Square sq;
               vector<int> minEls;
               for (int i = 0; i < variants.size(); i++) {

                    int minElement = variants[i].size();
                    minEls.push_back(minElement);
               }

               int minElementIndex =
std::min_element(minEls.begin(), minEls.end()) - minEls.begin();
               int minElement = *std::min_element(minEls.begin(),
minEls.end());

               std::cout << minElement << '\n';
```

```
                for (int i = variants.size()-1; i > 0; i--) {

                    if (minElement == variants[i].size())
                    {
                        for (int j = variants[i].size()-1; j >= 0;
j--) {

                            sq = variants[i][j];
                            variants[i].pop_back();
                            std::cout << sq.x * k + 1 << " " <<
sq.y * k + 1 << " " << sq.len * k;
                            if (!(variants[i].empty()))
                                std::cout << std::endl;
                        }
                    }
                    else
                        variants.pop_back();
                }




        }
    };

    int simplify(int size) {

        int start = size;
        for (int i = 2; i <= size; i++){
            if (size % i == 0)
                return i;
        }
        return  start;
    }
```

```
int main(){

	int n, choice;

	std::cout << "console: 0, console(debugging) : 1,
file(debugging) :2 " << std::endl;
	std::cin >> choice;


	std::cin >> n;
	int start_size = simplify(n);
	Matrix matrix(start_size, 0);
	int kf = n / start_size;

	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;
```

```
        }

        matrix.choice = choice;
        srand(time(0));

        if (choice == 0){

             matrix.backtracking(std::cout);
             matrix.print(kf);
        }
        else if (choice == 1) {

             matrix.backtracking(std::cout);
        }
        else if (choice == 2){

             std::ofstream file;
             file.open("result.txt");

             if (!file.is_open()) {
                  std::cout << "Incorrect!\n";
                  return 0;
             }
             matrix.backtracking(file);

        }

        cout << std::endl << "runtime = " << clock() / 1000.0 <<
endl;

        return 0;
```

```
}
```