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

ОТЧЕТ

по лабораторной работе №1
по дисциплине «Построение и анализ алгоритмов»
Тема: Поиск с возвратом
Вариант 2и

Студентка гр. 1381 _____ Рымарь М.И.

Преподаватель _____ Токарев А.П.

Санкт-Петербург
2023

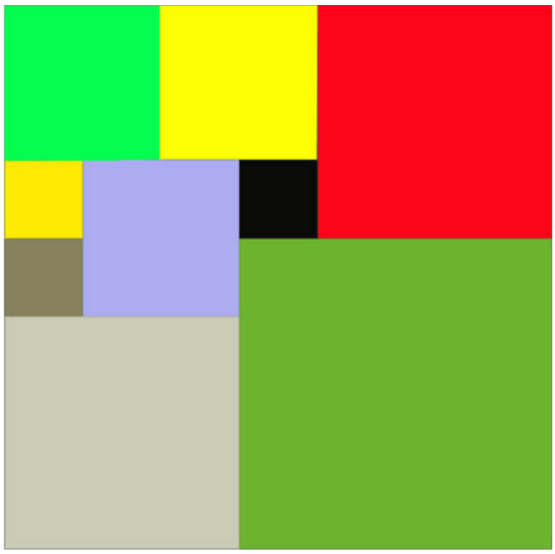
Цель работы.

Изучить алгоритм поиска с возвратом. Применить его на практике, выполнив задание лабораторной работы.

Задание.

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.

Входные данные: Размер столешницы - одно целое число N (2<=N<=20).

Выходные данные: Одно число K, задающее минимальное количество обрезков(квадратов), из которых можно построить столешницу(квадрат) заданного размера N. Далее должны идти K строк, каждая из которых должна содержать три целых числа x, y и w, задающие координаты левого верхнего угла (1<=x, y<=N) и длину стороны соответствующего обрезка(квадрата).

Индивидуализация (2и): Итеративный бэктрекинг. Исследование времени выполнения от размера квадрата.

Основные теоретические положения.

Поиск с возвратом, бэктрекинг — общий метод нахождения решений задачи, в которой требуется полный перебор всех возможных вариантов в некотором множестве. Как правило, позволяет решать задачи, в которых ставятся вопросы типа: «Перечислите все возможные варианты …», «Сколько существует способов …» и т.п.

Незначительные модификации метода поиска с возвратом, связанные с представлением данных или особенностями реализации, имеют и иные названия: метод ветвей и границ, поиск в глубину, метод проб и ошибок и т. д. Поиск с возвратом практически одновременно и независимо был изобретен многими исследователями ещё до его формального описания.

Описание метода: Решение задачи методом поиска с возвратом сводится к последовательному расширению частичного решения. Если на очередном шаге такое расширение провести не удается, то возвращаются к более короткому частичному решению и продолжают поиск дальше. Данный алгоритм позволяет найти все решения поставленной задачи, если они существуют. Для ускорения метода стараются вычисления организовать таким образом, чтобы как можно раньше выявлять заведомо неподходящие варианты. Зачастую это позволяет значительно уменьшить время нахождения решения.

Использование метода: Классическим примером использования алгоритма поиска с возвратом является задача о восьми ферзях. Её формулировка такова: «Расставить на стандартной 64-клеточной шахматной доске 8 ферзей так, чтобы ни один из них не находился под боем другого». Сперва на доску ставят одного ферзя, а потом пытаются поставить каждого следующего ферзя так, чтобы его не били уже установленные ферзи. Если на очередном шаге такую установку сделать нельзя — возвращаются на шаг назад и пытаются поставить ранее установленного ферзя на другое место.

Недостатки: Метод поиска с возвратом является универсальным. Достаточно легко проектировать и программировать алгоритмы решения задач с использованием этого метода. Однако время нахождения решения может быть очень велико даже при небольших размерностях задачи (количестве исходных данных), причём настолько велико (может составлять годы или даже века), что о практическом применении не может быть и речи. Поэтому при проектировании таких алгоритмов, обязательно нужно теоретически оценивать время их работы на конкретных данных. Существуют также задачи выбора, для решения которых можно построить уникальные, «быстрые» алгоритмы, позволяющие быстро получить решение даже при больших размерностях задачи. Метод поиска с возвратом в таких задачах применять неэффективно.

Выполнение работы.

В ходе выполнения лабораторной работы были реализованы две структуры: Square и Backtracking_head. Первая структура хранит информацию о параметрах маленьких квадратов (координаты левого верхнего угла и ширина). Вторая структура хранит информацию, необходимую для реализации алгоритма бэктрекинга (в том числе заполненность существующего квадрата, внесённые малые квадраты и их число).

Для обработки подающегося значения, были прописаны случаи, в которых число и параметры маленьких квадратов можно определить без использования алгоритма бэктрекинга — таким образом отдельно рассмотрены числа (размеры квадратов) кратные 2, кратные 3, кратные 5.

Для корректности работы алгоритма на составных числах, изначально находится наименьший простой делитель и уже с ним происходит обработка посредством бэктрекинга, далее полученные значения домножаются на сохранённый остаток от деления, что позволяет заполнить квадрат полностью наименьшим числом малых квадратов.

Функции, используемые в программе:

- New_Square() — добавляет новый квадрат на исходный, заполняя участок последнего по параметрам нового;

- Check() — проверка на возможность вмещения нового квадрата в исходный;

- Multiple2() — функция, которая передаёт вектор требуемых значений, введённых вручную для четных чисел;

- Multiple3() — функция, которая передаёт вектор требуемых значений, введённых вручную для чисел, кратных 3;

- Multiple5() — функция, которая передаёт вектор требуемых значений, введённых вручную для чисел, кратных 5;

- Backtracking() — Реализация алгоритма итеративного бэктрекинга: Заводится переменная, в которой хранится минимальное число квадратов, изначально задается результат как n+3+1, так как для простых чисел мы изначально фиксируем 3 квадрата, остальное пространство заполняем посредством перебора вариантов. Перебор основывается на реализации стека, хранящего возможные варианты, который итеративно дополняется (в случае необходимости) и обрабатывается;

- Output() — вывод требуемых значений в корректном формате.

Исходный код программы представлен в приложении А.

Тестирование.

Результаты тестирования представлены в таблице 1.

Ввод	Результат	Комментарий
3	6 1 1 1 1 2 1 1 3 1 3 1 1 2 2 2	Верно, простое число особый случай

	2 1 1 0.004	
6	4 1 1 3 1 4 3 4 1 3 4 4 3	Верно, составное число
7	9 1 1 2 1 3 2 3 1 2 3 3 1 3 4 1 4 3 1 4 4 4 5 1 3 1 5 3	Верно, простое число
29	14 1 1 8 1 9 7 8 9 2 8 11 5 9 1 7 9 8 1 10 8 3 13 8 3 13 11 3 13 14 2 15 14 1 15 15 15 16 1 14 1 16 14	Верно, простое большое число

Таблица 1 - Тестирование

Исследование алгоритма.

Зависимость времени работы алгоритма от размера квадрата (при простых числах, так как составные оптимизированы) представлены в таблице 2.

Значения	Время
7	0.008
11	0.013
13	0.026
17	0.15
23	2.603
29	24.768
37	108.936

Таблица 2 – Время работы алгоритма

На основании полученных данных был построен график. График представлен на рисунке 1, где ось x – размер квадрата, ось y – время работы алгоритма.

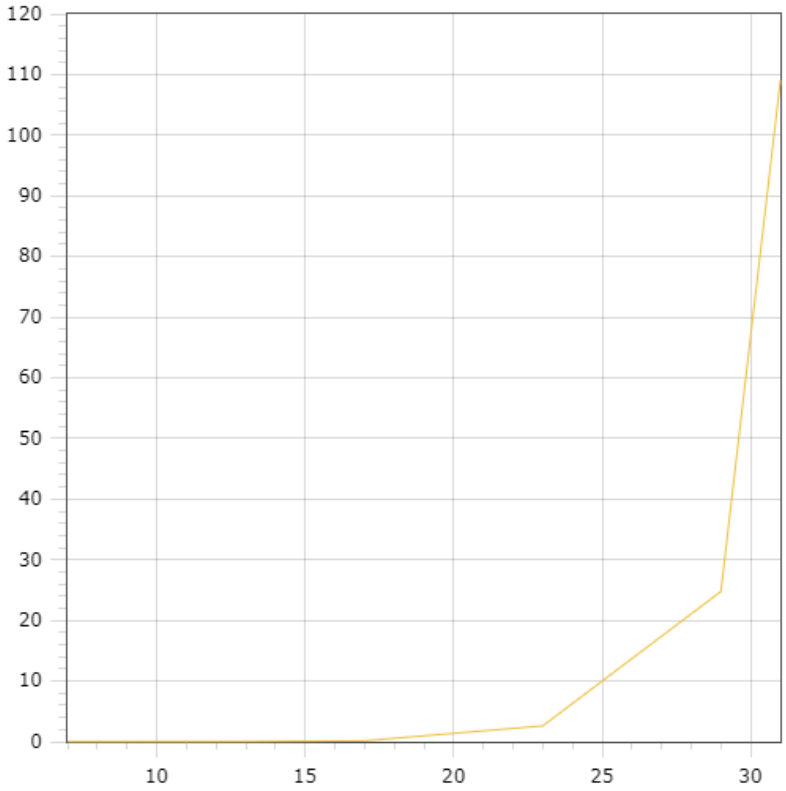


Рисунок 1 – График зависимости

Проанализировав полученный график, можно сделать вывод о том, что время работы алгоритма близка к факториальной зависимости.

Выводы.

В ходе выполнения лабораторной работы изучен поиск с возвратом. Алгоритм применён для решения задачи о разбиении квадрата на маленькие квадраты разных размеров без перекрытия и выхода за границы. Также выполнено дополнительное задание в виде исследования времени работы алгоритма от размера квадрата.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД ПРОГРАММ

```
#INCLUDE <VECTOR>
#INCLUDE <IOSTREAM>
#INCLUDE <STACK>

USING NAMESPACE STD;

STRUCT SQUARE{
    SQUARE(INT X = 0, INT Y = 0, INT WIDTH = 0){
        THIS->X = X;
        THIS->Y = Y;
        THIS->W = WIDTH;
    }
    INT X, Y, W;
};

STRUCT BACKTRACKING_HEAD{
    BACKTRACKING_HEAD(VECTOR<VECTOR<INT>>   BASE,   INT   COUNT,
VECTOR<SQUARE> SQUARES){
        THIS->BASE = BASE;
        THIS->SQUARES = SQUARES;
        THIS->COUNT = COUNT;
    }
    VECTOR<VECTOR<INT>> BASE;
    VECTOR<SQUARE> SQUARES;
    INT COUNT;
};

VOID NEW_SQUARE(VECTOR<VECTOR<INT>> &BASE, SQUARE SQUARE){
    FOR (INT I = SQUARE.X; I < SQUARE.X + SQUARE.W; I++){
        FOR (INT J = SQUARE.Y; J < SQUARE.Y + SQUARE.W; J++)
            BASE[I][J] = 1;
    }
}

BOOL CHECK(INT N, VECTOR<VECTOR<INT>> BASE, SQUARE SQUARE){   //
IF IT'S POSSIBLE TO PLACE A SQUARE
    BOOL FLAG = TRUE;
    IF (SQUARE.X + SQUARE.W >= N || SQUARE.Y + SQUARE.W >= N){
        FLAG = FALSE;
    }
    IF   (BASE[SQUARE.X][SQUARE.Y   +   SQUARE.W]   !=   0   ||
BASE[SQUARE.X + SQUARE.W][SQUARE.Y]){
        FLAG = FALSE;
    }
    IF (BASE[SQUARE.X + SQUARE.W][SQUARE.Y + SQUARE.W] != 0){
        FLAG = FALSE;
    }
    IF (SQUARE.X < 0 || SQUARE.Y < 0){
        FLAG = FALSE;
```

```
    }
    RETURN FLAG;
}

VOID MULTIPLE2(INT N, VECTOR<SQUARE> &ANSWER){
    ANSWER = {{0, 0, N/2},
              {0, N/2, N/2},
              {N/2, 0, N/2},
              {N/2, N/2, N/2}};
}

VOID MULTIPLE3(INT N, VECTOR<SQUARE> &ANSWER){
    ANSWER = {{0, 0, N/3},
              {0, N/3, N/3},
              {0, 2 * (N/3), N/3},
              {2 * (N/3), 0, N/3},
              {N/3, N/3, 2 * (N/3)},
              {N/3, 0, N/3}};
}

VOID MULTIPLE5(INT N, VECTOR<SQUARE> &ANSWER){
    ANSWER = {{0, 0, 3 * (N/5)},
              {0, 3 * (N/5), 2 * (N/5)},
              {2 * (N/5), 3 * (N/5), N/5},
              {2 * (N/5), 4 * (N/5), N/5},
              {3 * (N/5), 0, 2 * (N/5)},
              {3 * (N/5), 3 * (N/5), 2 * (N/5)},
              {3 * (N/5), 2 * (N/5), N/5},
              {4 * (N/5), 2 * (N/5), N/5},
              };
}

VOID BACKTRACKING(INT N, INT &COUNT, VECTOR<SQUARE> &ANSWER){
    VECTOR<VECTOR<INT>> BASE(N);
    FOR (INT I = 0; I < N; I++){
        BASE[I] = VECTOR<INT>(N, 0);
    }
    VECTOR<SQUARE> EXISTED_SQAURES;
    NEW_SQUARE(BASE, SQUARE(N/2, N/2, N/2 + 1));
    EXISTED_SQAURES.EMPLACE_BACK(N/2, N/2, N/2 + 1);
    NEW_SQUARE(BASE, SQUARE(N-N/2, 0, N/2));
    EXISTED_SQAURES.EMPLACE_BACK(N-N/2, 0, N/2);
    NEW_SQUARE(BASE, SQUARE(0, N-N/2, N/2));
    EXISTED_SQAURES.EMPLACE_BACK(0, N-N/2, N/2);

    STACK<BACKTRACKING_HEAD> STACK;
    STACK.EMPLACE(BASE, 0, ANSWER);
    WHILE (!STACK.EMPTY()){
        BACKTRACKING_HEAD LAST = STACK.TOP();
        STACK.POP();
        BOOL ADD_SQUARE = TRUE;
        PAIR<INT, INT> POINT = {-1, -1};
        FOR (INT X = 0; X < N/2 + 1; X++){
```

```
            FOR (INT Y = 0; Y < N/2 + 1; Y++){
                IF (LAST.BASE[X][Y] == 0){
                    POINT = {X, Y};
                    ADD_SQUARE = FALSE;
                    BREAK;
                }
            }
            IF (POINT.FIRST > -1){
                BREAK;
            }
        }
        IF (ADD_SQUARE) {
            ANSWER = LAST.SQUARES;
            COUNT = LAST.COUNT;
        }
        IF (COUNT <= LAST.COUNT + 1){
            CONTINUE;
        }
        FOR (INT W = 0; W < N - 1; W++){
            PAIR<INT, INT> START = {POINT.FIRST, POINT.SECOND};
            IF    (CHECK(N,    LAST.BASE,    SQUARE(POINT.FIRST,
POINT.SECOND, W))){
                FOR (INT X = START.FIRST; X <= POINT.FIRST + W;
X++){
                    LAST.BASE[X][POINT.SECOND     +     W]     =
LAST.SQUARES.SIZE() + 1;
                }
                FOR (INT Y = START.SECOND; Y <= POINT.SECOND +
W; Y++){
                    LAST.BASE[POINT.FIRST+W][Y]                =
LAST.SQUARES.SIZE()+1;
                }
                START.FIRST++;
                START.SECOND++;
                LAST.SQUARES.PUSH_BACK(SQUARE({POINT.FIRST,
POINT.SECOND, W+1}));
                STACK.EMPLACE(LAST.BASE,           LAST.COUNT+1,
LAST.SQUARES);
                LAST.SQUARES.POP_BACK();
            } ELSE BREAK;
        }
    }
    COPY(EXISTED_SQAURES.BEGIN(),          EXISTED_SQAURES.END(),
BACK_INSERTER(ANSWER));

}

VOID OUTPUT(VECTOR<SQUARE> ANSWER, INT OST){
    COUT << ANSWER.SIZE() << '\N';
    FOR (INT I = 0; I < ANSWER.SIZE(); I++){
        COUT << (ANSWER[I].X * OST + 1) << ' ' << (ANSWER[I].Y *
OST + 1) << ' ' << (ANSWER[I].W * OST) << '\N';
    }
```

```
}

INT MAIN(){
    INT N;
    CIN >> N;
    CLOCK_T START = CLOCK();
    VECTOR<SQUARE> ANSWER;
    INT COUNT = N + 3 + 1;
    INT PRIME = N;
    INT OST = 1;
    IF (N % 2 == 0){
        MULTIPLE2(N, ANSWER);
    } ELSE IF (N % 3 == 0){
        MULTIPLE3(N, ANSWER);
    } ELSE IF (N % 5 == 0){
        MULTIPLE5(N, ANSWER);
    } ELSE {
        FOR (INT I = 3; I < N; I++) {
            IF (N % I == 0) {
                PRIME = I;
                OST = N / I;
                BREAK;
            }
        }
        BACKTRACKING(PRIME, COUNT, ANSWER);
    }
    OUTPUT(ANSWER, OST);
    CLOCK_T END = CLOCK();
    COUT << '\N' << (FLOAT)(END-START)/CLOCKS_PER_SEC << '\N';
    RETURN 0;
}
```