**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**

Студент гр. 7304 _____ Овчинников Н.В.

Преподаватель _____ Филатов А.Ю.

Санкт-Петербург
2019

# Цель работы

Ознакомиться с алгоритмом поиска с возвратом, получить навыки его программирования и применения на языке программирования C++.

# Задание

Разработать программу, которая делит квадрат со стороной N на квадраты со сторонами не более, чем N-1. Количество получившихся квадратов должно быть наименьшим из всех возможных вариантов разбиения.

# Основные теоретические положения

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

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.

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.

# Экспериментальные результаты

1. Создал необходимые для работы структуры: qdr (значения квадрата и указатель на следующий), headPointer (указатель на первый квадрат), answer (хранит в себе промежуточное и лучшее количество квадратов на столе, промежуточный и лучший списки, в которых хранятся состояния стола, и размер стола N).

```
typedef struct qdr {
    short x;
    short y;
    short w;
    qdr *next;

    void initQdr(short xx, short yy, short ww){
        x = xx;
        y = yy;
        w = ww;
    }
} qdr;

typedef struct headPointer {
    qdr *pointer;
} headPointer;

typedef struct answer {
    short bestValue;
    short singleValue;
    headPointer *bestArr;
    headPointer *singleArr;
    short N;

    answer(){
        bestValue = 1000;
        singleValue = 0;
        bestArr = (headPointer*)malloc(sizeof(headPointer));
        bestArr->pointer = NULL;
        singleArr = (headPointer*)malloc(sizeof(headPointer));
        singleArr->pointer = NULL;
        N = 0;
    }

    ~answer(){
        free(bestArr);
        free(singleArr);
    }

    void cleanBestArray(){
        if(bestArr->pointer == NULL)
            return;
        qdr *tmp1 = bestArr->pointer;
        qdr *tmp2 = bestArr->pointer;
        while(tmp1){
            tmp1 = tmp1->next;
            free(tmp2);
            tmp2 = tmp1;
        }
        bestArr->pointer = NULL;
    }
} answer;
```

2. Реализовал функции для работы со списком: вставка в конец списка, удаление из конца списка, поиск последнего элемента в списке, печать содержимого элементов списка (qdr):

```
qdr *findEndOfList(headPointer *head){
    if(head->pointer == NULL)
        return NULL;
    qdr *tmp = head->pointer;
    while(tmp->next)
        tmp = tmp->next;
    return tmp;
}

//вставка в конец списка
void insertQdrToList(headPointer *head, short x, short y, short w){
    qdr *elemForInsert = (qdr*)calloc(1, sizeof(qdr));
    elemForInsert->initQdr(x, y, w);
    if(head->pointer == NULL){
        head->pointer = elemForInsert;
        return;
    }

    qdr *tmp = head->pointer;
    while(tmp->next){
        tmp = tmp->next;
    }
    tmp->next = elemForInsert;
}

//удаление из последнего элемента из списка
void removeQdrFormList(headPointer *head){
    if(head->pointer == NULL)
        return;
    if(head->pointer->next == NULL){
        free(head->pointer);
        head->pointer = NULL;
        return;
    }
    qdr *tmp = head->pointer;
    while(tmp->next->next){
        tmp = tmp->next;
    }
    free(tmp->next);
    tmp->next = NULL;
}

void printList(headPointer *head){
    qdr *tmp = head->pointer;
    while(tmp){
        printf("%hd %hd %hd\n", tmp->x+1, tmp->y+1, tmp->w);
        tmp = tmp->next;
    }
}
```

3. Реализовал функции для работы со столом, на котором располагаются квадраты (на столе: 0 – пусто, 1 – занято):

```
short getFirstEmpty(unsigned char **table, short &x, short &y, short N){
    for(short i=0; i<N; i++){
        for(short k=0; k<N; k++){
```

```
            if(table[i][k]==0){
                x=k;
                y=i;
                return 0;
            }
        }
    }
    return 1;
}

//поставить квадрат
void putQdr(unsigned char **table, short x, short y, short w, answer *ans,
unsigned char val){
    if(table[y][x]==1 || x+w > ans->N || y+w > ans->N)
        return;

    for(short t=x; t<x+w; t++)
        if(table[y][t]==1)
            return;

    for(short i=y;i<y+w;i++)
        for(short k=x; k<x+w; k++)
            table[i][k]=val;
    ans->singleValue++;
    insertQdrToList(ans->singleArr, x, y, w);
//printTable(table, ans->N);
}

//удалить последний квадрат из singleArr
//вернуть 1 если нечего удалять, 0 - удален квадрат
inline short removeQdr(unsigned char **table, answer *ans, qdr &prevQdr){
    if(ans->singleArr->pointer == NULL)
        return 1;
    qdr *tmp = ans->singleArr->pointer;
    while(tmp->next)
        tmp = tmp->next;
    if(table[tmp->y][tmp->x]==2)
        return 2;
    for(short i=tmp->y; i < tmp->y + tmp->w; i++)
        for(short k=tmp->x; k<tmp->x + tmp->w; k++)
            table[i][k]=0;
    prevQdr.initQdr(tmp->x, tmp->y, tmp->w);
    removeQdrFormList(ans->singleArr);
    ans->singleValue--;
//printTable(table, ans->N);
    return 0;
}
```

4. Реализовал главную рекурсивную функцию, которая пробирает все варианты для заданного N (<23) и выбирает расстановку с наименьшим числом квадратов:

```
inline void go(unsigned char **table, answer *ans, short &key){
    if(key)
        return;
    short x=0, y=0;
    if(getFirstEmpty(table, x, y, ans->N) || ans->singleValue >= ans-
>bestValue){
        if(ans->singleValue < ans->bestValue){
            ans->bestValue = ans->singleValue;
            copySingleArrInBestArr(ans);
```

```
        }
        qdr prevQdr;
        key = removeQdr(table, ans, prevQdr);
        short size = ans->N > 20 ? 3 : 2;
        size = ans->N > 24 ? 4 : size;
        while(prevQdr.w < size && key==0){
            key = removeQdr(table, ans, prevQdr);
        }
        if(key)
            return;
        putQdr(table, prevQdr.x, prevQdr.y, prevQdr.w-1, ans, 1);
        go(table, ans, key);
    }
    else {
        for(short i=ans->N/2/2+1; i>=1; i--)
           putQdr(table, x, y, i, ans, 1);
        go(table, ans, key);
    }
    return;
}
```

Также учел ситуации, когда размер стола кратен двум и трём, при помощи функции begin:

```
short begin(unsigned char **table, answer *ans){
    if(ans->N % 2 == 0){
        for(short i=0; i<ans->N; i += ans->N/2)
            for(short k=0; k<ans->N; k += ans->N/2)
                putQdr(table, k, i, ans->N/2, ans, 2);
        ans->bestValue = 4;
        copySingleArrInBestArr(ans);
        return 2;
    }
    if(ans->N % 3 == 0 && ans->N != 3){
        putQdr(table, 0, 0, ans->N/3*2, ans, 2);
        putQdr(table, ans->N/3*2, 0, ans->N/3, ans, 2);
        putQdr(table, ans->N/3*2, ans->N/3, ans->N/3, ans, 2);
        putQdr(table, ans->N/3*2, ans->N/3*2, ans->N/3, ans, 2);
        putQdr(table, 0, ans->N/3*2, ans->N/3, ans, 2);
        putQdr(table, ans->N/3, ans->N/3*2, ans->N/3, ans, 2);
        ans->bestValue = 6;
        copySingleArrInBestArr(ans);
        return 3;
    }
    putQdr(table, 0, 0, ans->N/2 + 1, ans, 2);
    putQdr(table, ans->N/2+1, 0, ans->N/2, ans, 2);
    putQdr(table, 0, ans->N/2+1, ans->N/2, ans, 2);
    ans->singleValue = 3;
    return 0;
}
```

# Вывод

В ходе выполнения лабораторной работы ознакомился с алгоритмом поиска с возвратом, а также сумел наглядно продемонстрировать его на примере поставленной задачи.

# Приложение: исходный код программы

```
#include <stdlib.h>
#include <stdio.h>

typedef struct qdr {
   short x;
   short y;
   short w;
   qdr *next;

   void initQdr(short xx, short yy, short ww){
      x = xx;
      y = yy;
      w = ww;
   }
} qdr;

typedef struct headPointer {
   qdr *pointer;
} headPointer;

typedef struct answer {
   short bestValue;
   short singleValue;
   headPointer *bestArr;
   headPointer *singleArr;
   short N;

   answer(){
      bestValue = 1000;
      singleValue = 0;
      bestArr = (headPointer*)malloc(sizeof(headPointer));
      bestArr->pointer = NULL;
      singleArr = (headPointer*)malloc(sizeof(headPointer));
      singleArr->pointer = NULL;
      N = 0;
   }

   ~answer(){
      free(bestArr);
      free(singleArr);
   }

   void cleanBestArray(){
      if(bestArr->pointer == NULL)
         return;
      qdr *tmp1 = bestArr->pointer;
      qdr *tmp2 = bestArr->pointer;
      while(tmp1){
         tmp1 = tmp1->next;
         free(tmp2);
```

```
            tmp2 = tmp1;
        }
        bestArr->pointer = NULL;
    }
} answer;

//инициализация стола
unsigned char **initTable(short N){
    unsigned char **table = (unsigned char**)malloc(N * sizeof(unsigned char*));
    for(short i=0;i<N;i++)
        table[i] = (unsigned char*)calloc(N, sizeof(unsigned char));
    return table;
}

//удаление стола
void removeTable(unsigned char **table, short N){
    for(short i=0; i<N; i++)
        free(table[i]);
    free(table);
}

//возврат 1, если стол заполнен полностью
short getFirstEmpty(unsigned char **table, short &x, short &y, short N){
    for(short i=0; i<N; i++){
        for(short k=0; k<N; k++){
            if(table[i][k]==0){
                x=k;
                y=i;
                return 0;
            }
        }
    }
    return 1;
}

qdr *findEndOfList(headPointer *head){
    if(head->pointer == NULL)
        return NULL;
    qdr *tmp = head->pointer;
    while(tmp->next)
        tmp = tmp->next;
    return tmp;
}

//вставка в конец списка
void insertQdrToList(headPointer *head, short x, short y, short w){
    qdr *elemForInsert = (qdr*)calloc(1, sizeof(qdr));
    elemForInsert->initQdr(x, y, w);
    if(head->pointer == NULL){
        head->pointer = elemForInsert;
        return;
    }
```

```
    qdr *tmp = head->pointer;
    while(tmp->next){
        tmp = tmp->next;
    }
    tmp->next = elemForInsert;
}

//удаление из последнего элемента из списка
void removeQdrFormList(headPointer *head){
    if(head->pointer == NULL)
        return;
    if(head->pointer->next == NULL){
        free(head->pointer);
        head->pointer = NULL;
        return;
    }
    qdr *tmp = head->pointer;
    while(tmp->next->next){
        tmp = tmp->next;
    }
    free(tmp->next);
    tmp->next = NULL;
}

void printList(headPointer *head){
    qdr *tmp = head->pointer;
    while(tmp){
        printf("%hd %hd %hd\n", tmp->x+1, tmp->y+1, tmp->w);
        tmp = tmp->next;
    }
}

void copySingleArrInBestArr(answer *ans){
    ans->cleanBestArray();
    qdr *tmp = ans->singleArr->pointer;
    while(tmp){
        insertQdrToList(ans->bestArr, tmp->x, tmp->y, tmp->w);
        tmp = tmp->next;
    }
}

//поставить квадрат
void putQdr(unsigned char **table, short x, short y, short w, answer *ans, unsigned char val){
    if(table[y][x]==1 || x+w > ans->N || y+w > ans->N)
        return;

    for(short t=x; t<x+w; t++)
        if(table[y][t]==1)
            return;

    for(short i=y;i<y+w;i++)
        for(short k=x; k<x+w; k++)
            table[i][k]=val;
```

```
    ans->singleValue++;
    insertQdrToList(ans->singleArr, x, y, w);
}

//удалить последний квадрат из singleArr
//вернуть 1 если нечего удалять, 0 - удален квадрат
inline short removeQdr(unsigned char **table, answer *ans, qdr &prevQdr){
    if(ans->singleArr->pointer == NULL)
        return 1;
    qdr *tmp = ans->singleArr->pointer;
    while(tmp->next)
        tmp = tmp->next;
    if(table[tmp->y][tmp->x]==2)
        return 2;
    for(short i=tmp->y; i < tmp->y + tmp->w; i++)
        for(short k=tmp->x; k<tmp->x + tmp->w; k++)
            table[i][k]=0;
    prevQdr.initQdr(tmp->x, tmp->y, tmp->w);
    removeQdrFormList(ans->singleArr);
    ans->singleValue--;
    return 0;
}

inline void go(unsigned char **table, answer *ans, short &key){
    if(key)
        return;
    short x=0, y=0;
    if(getFirstEmpty(table, x, y, ans->N) || ans->singleValue >= ans->bestValue){
        if(ans->singleValue < ans->bestValue){
            ans->bestValue = ans->singleValue;
            copySingleArrInBestArr(ans);
        }
        qdr prevQdr;
        key = removeQdr(table, ans, prevQdr);
        short size = ans->N > 20 ? 3 : 2;
        size = ans->N > 24 ? 4 : size;
        while(prevQdr.w < size && key==0){
            key = removeQdr(table, ans, prevQdr);
        }
        if(key)
            return;
        putQdr(table, prevQdr.x, prevQdr.y, prevQdr.w-1, ans, 1);
        go(table, ans, key);
    }
    else {
        for(short i=ans->N/2/2+1; i>=1; i--)
            putQdr(table, x, y, i, ans, 1);
        go(table, ans, key);
    }
    return;
}

short begin(unsigned char **table, answer *ans){
```

```
    if(ans->N % 2 == 0){
        for(short i=0; i<ans->N; i += ans->N/2)
            for(short k=0; k<ans->N; k += ans->N/2)
                putQdr(table, k, i, ans->N/2, ans, 2);
        ans->bestValue = 4;
        copySingleArrInBestArr(ans);
        return 2;
    }
    if(ans->N % 3 == 0 && ans->N != 3){
        putQdr(table, 0, 0, ans->N/3*2, ans, 2);
        putQdr(table, ans->N/3*2, 0, ans->N/3, ans, 2);
        putQdr(table, ans->N/3*2, ans->N/3, ans->N/3, ans, 2);
        putQdr(table, ans->N/3*2, ans->N/3*2, ans->N/3, ans, 2);
        putQdr(table, 0, ans->N/3*2, ans->N/3, ans, 2);
        putQdr(table, ans->N/3, ans->N/3*2, ans->N/3, ans, 2);
        ans->bestValue = 6;
        copySingleArrInBestArr(ans);
        return 3;
    }
    putQdr(table, 0, 0, ans->N/2 + 1, ans, 2);
    putQdr(table, ans->N/2+1, 0, ans->N/2, ans, 2);
    putQdr(table, 0, ans->N/2+1, ans->N/2, ans, 2);
    ans->singleValue = 3;
    return 0;
}

int main()
{
    answer ans;
    scanf("%hd", &(ans.N));
    unsigned char **table = initTable(ans.N);   //стол
    short key = 0;  //ключ завершения работы

    short start = begin(table, &ans);
    if(start!=2 && start!=3)
        go(table, &ans, key);


    printf("%hd\n", ans.bestValue);
    printList(ans.bestArr);

    removeTable(table, ans.N);
    return 0;
}
```