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 C++.

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).

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 $2 \leq N \leq 20$.

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 ($1 \leq x,y \leq N$) и длину стороны соответствующего обрезка(квадрата).

Описание алгоритма

В начале программы проверяется делимость размера квадрата на 2 и 3. Если размер делится на 2, то столешница разбивается на 4 квадрата со сторонами N/2. Если размер делится на 3, то столешница разбивается на 6 квадратов: один квадрат со стороной N*2/3, а остальные пять со сторонами N/3.

Если размер квадрата не делится ни на 2, ни на 3, то сначала выставляются три квадрата (N/2+1, N/2, N/2) и вызывается рекурсивная функция, которая ищет первую свободную клетку на столе и ставит в ней квадрат, начиная с самого большого возможного и заканчивая квадратом 1x1.

Если функции удалось поставить квадрат, то она вызывается рекурсивно до тех пор, пока весь стол не будет заполнен, либо пока текущее количество квадратов на столе не превысит (или не станет равным) лучшему количеству квадратов в предыдущих расстановках.

Когда такое произойдёт, со стола либо будет удалён один последний поставленный квадрат, и на его место встанет квадрат с размер меньше на единицу, либо со стола будут удаляться последний поставленные квадраты, размер которых равен 1. Как только функция удалит квадрат со стороной большей 1, снова вызовется рекурсивная функция с той же позиции.

Функция завершит свою работу, когда будут рассмотрены все варианты расстановок квадратов.

Ход работы

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 (<20) 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;
}
```