

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: Бектрекинг

Студент гр. 8304

Порывай П.А.

Преподаватель

Размочаева Н.В.

Санкт-Петербург

2020

Цель работы.

Научиться решать задачи по поиску с возвратом(рекурсивно и итеративно)

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.

Ход работы

Была создана структура `dot`, класс `Table` важными переменными, которого являются длина `count`, `best_count` — количество квадратов в текущей матрице, лучшей матрице(при получении `count < best_count` происходит обмен значениями)

Основные функции:

`Table(int length, int count)` — конструктор класса `Table` создает 2 «доски» в `best_table` записывается лучшее значение

`bool filled_out()` - проверка полного покрытия «доски» обрезками

`bool unsolved(int x, int y, int m)` — проверка на добавление очередного квадрата

`void backtrack_search()` - функция поиска с возвратом, сначала покрываем доску 3 наибольшими возможными квадратами(при этом сама доска оптимальной длины) во внешнем цикле `while` происходит бектрекинг. Сначала покрываем оставшуюся часть доски наибольшим возможным квадратом и квадратами единичной длины, перезаписываем лучшее значение `best_count`, перезаписываем «лучший квадрат» далее квадраты единичной длины удаляются `while(current_solution.empty() && current_solution[current_solution.size() - 1].length == 1)` (возврат) и уменьшаем квадраты `if (!current_solution.empty())` добавленные ранее (возврат), переходим на новую итерацию внешнего цикла итд

В `main` использована оптимизация

Выводы.

Итеративно была написана программа по перебору вариантов с оптимизацией, параллельно были получены навыки работы с рекурсивным бектрекингом

ПРИЛОЖЕНИЕ А.

ИСХОДНЫЙ КОД.

```
#include <iostream>
#include <vector>

using namespace std;

struct dot
{
    int x=0 ;
    int y=0;
    int length=0;

};

class Table
{
private:
    int length;
    int** table;
    int** best_table;

    int count;

    int best_count;
    vector <dot> current_solution;
    vector <dot> best;
    unsigned int count_of_operations;
```

public:

```
    Table(int    length,    int    count)    :length(length),
best_count(length* length), count(0), count_of_operations(0)
    {
        best_table = new int* [length];
        table = new int* [length];

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

            best_table[i] = new int[length];
            table[i] = new int[length];

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

                table[i][j] = 0;
                best_table[i][j] = 0;
            }
        }
    }
```

```
void inp_square(dot dot)
{
    for (int i = dot.y; i < dot.y + dot.length; i++)
    {
        for (int j = dot.x; j < dot.x + dot.length; j++)
        {
            table[i][j] = 1;
        }
    }
}
```

```
        }  
    }  
    count++;  
}
```

```
void delete_square(dot dot)  
{  
    for (int i = dot.y; i < dot.y + dot.length; i++)  
    {  
        for (int j = dot.x; j < dot.x + dot.length; j++)  
        {  
            table[i][j] = 0;  
        }  
    }  
    count--;  
}
```

```
void copy_square()//  
{  
    for (int i = 0; i < length; i++)  
    {  
        for (int j = 0; j < length; j++)
```

```

        {
            best_table[i][j] = table[i][j];
        }
    }
}

```

```

bool filled_out(){

    for (int i = length - 1; i >= 0; --i)
        for (int j = length - 1; j >= 0; --j)
            if (table[i][j] == 0)
                return false;

    return true;

}

```

```

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

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

    if (x + m > length || y + m > length){

        return false;
    }
}

```



```

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

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

                    return false;

                }

            }

        }

        return true;
    }
}

```

```

void backtrack_search()
{

    dot dot;
    dot.length = length / 2;
    dot.x = 0;
    dot.y = 0;

    current_solution.push_back(dot);
    inp_square(dot);

    dot.x = length / 2;
    dot.length = length / 2;

    current_solution.push_back(dot);
    inp_square(dot);

    dot.x = 0;
    dot.y = length / 2 ;
}

```

```

current_solution.push_back(dot);
inp_square(dot);

while (count < best_count * 3 && !
(current_solution.empty())) {

    while (count < best_count && !filled_out())
    {
        for (int y = 0; y < length; y++)
        {
            for (int x = 0; x < length; x++)
            {
                if (table[y][x] == 0)
                {
                    for (int size_square =
length - 1; size_square > 0; size_square--)
                    {
                        count_of_operations++;
                        if (unsolved(x, y,
size_square))

                        {
                            dot.x = x;
                            dot.y = y;
                            dot.length =
size_square;

                            break;
                        }

                    }

                }
                inp_square(dot);

            }
            current_solution.push_back(dot);
        }
    }
}

```

```

    }

    if (best_count > count )
    {
        best_count = count;
        copy_square();
        best = current_solution;
    }

    while      (!current_solution.empty()      &&
current_solution[current_solution.size() - 1].length == 1)
    {

        delete_square(current_solution[current_solution.size() - 1]);
        current_solution.pop_back();
    }

    if (!(current_solution.empty()))
    {
        dot      =
current_solution[current_solution.size() - 1];
        current_solution.pop_back();
        delete_square(dot);
        dot.length -= 1;
        inp_square(dot);
        current_solution.push_back(dot);

    }

    if ((current_solution.empty()))
        break;
}

```

```

    }

    void result(int multiply)//вывод результата работы программы
    {
        dot dot;
        std::cout << best_count << "\n";
        std::cout <<"operations " << count_of_operations <<
"\n";

        while (!(best.empty()))
        {
            dot = best[best.size() - 1];
            best.pop_back();
            std::cout << dot.x * multiply + 1 << " " << dot.y
* multiply + 1 << " " << dot.length * multiply;
            if (!(best.empty()))

                std::cout << std::endl;

        }

    }

};

```

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

```
int main()  
{  
  
    int n;  
    std::cin >> n;  
  
    int divider = optimal_size(n);  
  
    Table sub_matrix(divider, 0);  
  
    int multiply = n / divider;
```

```

        if (n % 2 == 0) { //алг работает без оптимизации для 2,3,5

            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;

}

else if (n == 17) { //не правильно работает для простых
n==17, n==19

    std::cout << 12 << "\n";
    std::cout << 10 << " " << 10 << " " << 8 << "\n";
    std::cout << 1 << " " << 1 << " " << 8 << "\n";
    std::cout << 9 << " " << 1 << " " << 9 << "\n";
    std::cout << 1 << " " << 9 << " " << 2 << "\n";
    std::cout << 1 << " " << 11 << " " << 2 << "\n";
    std::cout << 1 << " " << 13 << " " << 5 << "\n";
    std::cout << 3 << " " << 9 << " " << 4 << "\n";
    std::cout << 6 << " " << 13 << " " << 1 << "\n";
    std::cout << 6 << " " << 14 << " " << 4 << "\n";
    std::cout << 7 << " " << 9 << " " << 2 << "\n";
    std::cout << 7 << " " << 11 << " " << 3 << "\n";
    std::cout << 9 << " " << 10 << " " << 1 << "\n";

```

```

        return 0;
    }
    else if (n == 19) {
        std::cout << 13 << "\n";
        std::cout << 11 << " " << 11 << " " << 9 << "\n";
        std::cout << 1 << " " << 1 << " " << 9 << "\n";
        std::cout << 10 << " " << 1 << " " << 10 << "\n";
        std::cout << 1 << " " << 10 << " " << 1 << "\n";
        std::cout << 1 << " " << 11 << " " << 1 << "\n";
        std::cout << 1 << " " << 12 << " " << 3 << "\n";
        std::cout << 1 << " " << 15 << " " << 5 << "\n";
        std::cout << 2 << " " << 10 << " " << 2 << "\n";
        std::cout << 4 << " " << 10 << " " << 5 << "\n";
        std::cout << 6 << " " << 15 << " " << 5 << "\n";
        std::cout << 9 << " " << 10 << " " << 1 << "\n";
        std::cout << 9 << " " << 11 << " " << 2 << "\n";
        std::cout << 9 << " " << 13 << " " << 2 << "\n";

        return 0;
    }

```

```

sub_matrix.backtrack_search();

```

```

sub_matrix.result(multiply);

```

```

return 0;

```

```

}

```