Московский авиационный институт
(национальный исследовательский университет)

**Факультет информационных технологий и прикладной математики**

**Кафедра вычислительной математики и программирования**

**Лабораторная работа №7 по курсу «Дискретный анализ»**


| | |
|---|---|
| Студент: | А. О. Ларченко |
| Преподаватель: | А. А. Кухтичев |
| Группа: | М8О-206Б |
| Дата: | |
| Оценка: | |
| Подпись: | |


**Москва, 2024**

# Лабораторная работа №7

**Задача:** Имеется натуральное число n. За один ход с ним можно произвести следующие действия:
- Вычесть единицу
- Разделить на два
- Разделить на три

При этом стоимость каждой операции – текущее значение n. Стоимость преобразования - суммарная стоимость всех операций в преобразовании. Вам необходимо с помощью последовательностей указанных операций преобразовать число n в единицу таким образом, чтобы стоимость преобразования была наименьшей. Делить можно только нацело.

**Форма ввода** В первой строке задано $2 <= n < 10^7$.

**Форма вывода** Числа в диапазоне от 0 до $2^{64}$-1 .

# 1 Описание

Эта задача решается методом динамического программирования, который позволяет решать задачи, комбинируя решения вспомогательных задач.

Динамическое программирование находит применение тогда, когда вспомогательные задачи не являются независимыми, т.е. когда разные вспомогательные задачи используют решения одних и тех же подзадач. В этом смысле алгоритм разбиения, многократно решая задачи одних и тех же типов, выполняет больше действий, чем было бы необходимо. В алгоритме динамического программирования каждая вспомогательная задача решается только один раз, после чего ответ сохраняется в таблице. то позволяет избежать одних и тех же повторных вычислений каждый раз, когда встречается данная подзадача.[1]

Пусть dp[i] - стоимость i-ой операции. Тогда dp[i] = i + min(dp[i+1], dp[i*2], dp[i*3]). Для решения задачи нам пригодится структура очередь. При этом стоит учесть,что мы идем от заданного числа n до 1, следовательно, чтобе не перебирать все возможные варианты на нашем отрезке длинной n, на каждом шаге мы будем сдвигаться проверять значения, которые стоят на позициях, dp[i-1], dp[i/2], dp[i/3] может возникнуть несклько случаев:
1) Эта позиция не иницилизирована. В таком случае записываем получившееся в значение в клетку, и добавляем индекс в очередь;
2) Эта позиция иницилизирована, её значение > нашего значения. В таком случае обновляем значение ячейки и добавляем индекс в очередь.
3) Эта позиция иницилизирована, её значение <= нашего значения. В таком случае мы пропускаем данную ячейку.

Начинаем наш алгоритм с того, что добавляем индекс n в очередь. И выполняем все итерации пока очередь не пуста. На кажом шаге мы выполняем действия, описанные выше.

Наивное решение: во-первых, не предусматривает мемоизации, а во-вторых, основывается на рекурсивном переборе всех возможных вариантов получения нужного числа. Такой метод имеет экспонециальную сложность ($O(3^2)$). Метод же динамического программирования, в свою очередь, позволяет нам избежать лишних вычислений. Итоговая сложность - O(N). Что выглядить довольно эффективно, по сравнению с экспонециальной сложностью.

# 2 Исходный код

Задача на ДП не является особо сложной, по сравнению с прошлыми лабами (65 строк !!!! по сравнению с 800...). Код получился настолько без заморочек, что я реализовал всю логику в функции *main*().

```
int main(){
    uint64_t n;
    cin>>n;
    vector<pair<uint64_t, uint64_t>> v(n+1); // <cmd, prev_sum>
    queue<uint64_t> q;
    q.push(n);
    while(q.size()>0){
        uint64_t a = q.front();
        q.pop();
        uint64_t s = v[a].second +a;
        vector<uint64_t> steps(3);
        steps[0] = a-1;
        if (a%2==0) steps[1] = a/2;
        if (a%3==0) steps[2] = a/3;
        for(int i=0; i<3;++i){
            if (steps[i]>0){
                if (v[steps[i]].second ==0){
                    v[steps[i]]=make_pair(i+1, s);
                    q.push(steps[i]);
                } else if (v[steps[i]].second > s){
                    v[steps[i]] = make_pair(i+1, s);
                    q.push(steps[i]);
                }
            }
        }
    }
    cout<<v[1].second<<'\n';
    stack<uint64_t> answ;
    answ.push(v[1].first);
    uint64_t cur_s = 1;
    while(true){
        uint64_t step = answ.top();
        if (step ==1){
            cur_s++;
        } else if(step==2){
            cur_s =cur_s*2;
        } else{
            cur_s =cur_s*3;
        }
        if (cur_s==n) break;
        answ.push(v[cur_s].first);
    }
    while(!answ.empty()){
```

```
        uint64_t a = answ.top();
        answ.pop();
        if(a ==1){
            cout<<"-1 ";
        } else {
            cout<<"/"<<a<<" ";
        }
    }
}
```

Как уже было сказано выше, вначале мы создаем массив длинной n, в который будем записывать минимальные суммы для этого числа, а также предыдущую итерацию, и ещё мы добавляем наш искомый по усслoвию элемент в очередь, затем заупскаем цикл, пока очередь будет не пуста. На каждой итерации этого цикла мы достаем 1 число, применяем к нему действия, описанные в условии ("$-1$ "/2 "/3") и смотрим на полученное число, если соответсвующая позиция массива не иницилизировано, просто записываем текущую сумму, если же она иницилизирована, то сравниваем со значением, стоящем на этом месте, если оно больше нашей текущей суммы, то записываем, в противном случае пропускаем эту итерацию. В первых двух случаях мы добавляем получившееся число в очередь.

# 3 Консоль

```
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/lab_7$ ./lab_7
82
202
-1 /3 /3 /3 /3
```

# 4 Тест производительности

В процессе тестирования будем сравнивать наш алгоритм ДП с наивным алгоритмом. Для теста возьмем такие числа: 79(простое), 82(из примера, делится на 2), 81(делится на 3), а кроме того, возьмем число побольше(503).

```
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/lab_7$ g++ banchmark.cpp -o banchmark
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/lab_7$ ./banchmark
82
Dummy algorithm : 0.010516243 s
My algorithm : 0.000018321 s
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/lab_7$ ./banchmark
79
Dummy algorithm : 0.009633805 s
My algorithm : 0.000031339 s
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/lab_7$ ./banchmark
81
Dummy algorithm : 0.010220480 s
My algorithm : 0.000017536 s
arsenii@PC-Larcha14:~/Documents/C_pp_uk/DA/lab_7$ ./banchmark
503
Dummy algorithm : 48.737464812 s
My algorithm : 0.000088194 s
```

Как можно заметить, наивный алгорим сокрушительно проигрывает абсолютно на всех тестах. Последний тест показывает экспонециальную сложность наивного алгоритма.

# 5 Выводы

Выполнив данную лабораторную работу, я пропитался силой динамического программирования и на реальном примере увидел насколько он оптимизирует алгоритмы. Но к сожалению, этот мощный метод применим не ко всех задачам, а только к задачам, в которых искомый ответ состоит из частей, каждая из которых в свою очередь дает оптимальное решение некоторой подзадачи.

# Список литературы


[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. Алгоритмы: построение и анализ, 2-е издание. — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))