ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«Динамическое программирование»

Выполнил работу

Тиганов Вадим

Академическая группа J3112

Принято

Дунаев Максим

Санкт-Петербург

2024

Структура отчёта:

1. Введение

Цель: изучить и применить на практике использование динамического программирования.

Задачи:

1. Выбрал задачу hard, поэтому решаю ее на leetcode.

2. Анализирую возможные подходы и решение с помощью ДП.

3. Провожу анализ написанного решения, асимптотики и понимаю работу алгоритма.

2. Теоретическая подготовка

Подход динамического программирования заключается в разбиении задачи на более мелкие подзадачи. В ходе работы алгоритм будет многократно решать их, храня промежуточные результаты. Отсюда вытекают и большие затраты по памяти.

3. Реализация

Полный код см. в приложении А.

Задача заключается в том, чтобы за минимальное количество операций превратить битовое представление числа в ноль.

Доступны две операции: превращаем самый правый бит в ноль или превращаем бит в ноль, если правый от него равен единице, а все следующие — нули.

```
vector<int> dp(31,0);
```

Инициализировал вектор для динамического программирования длины 31, так как это вместит любое число типа int.

```
for (int i = 1; i < 31; i++) {
    dp[i] = dp[i - 1] * 2 + 1;
}
```

Для начала заполняю массив такими данными. Показываем, сколько операций нужно для перевода каждого из 32 битов в ноль, а именно — dp[i]=dp[i-1]*2+1 операция для i-того бита. Почему — потому что для каждого следующего бита существует в два раза больше операций, а для перевода его самого нужна еще одна дополнительная.

```
int steps=0;
        int sign=1;
        for(int i = 30;i ⩾ 0; i--){
            if(n & (1 << i)){
                steps += dp[i] * sign;
                sign = 0 - sign;
            }

        }
```

Сам алгоритм считает количество операций так: смотрим, объявлен ли бит (=1) с помощью бинарной конструкции (n & (1 « i)) n — наше число. (Идем с конца)

Если объявлен — прибавляем количество операций для него, умноженное на знак. Знак нужен для корректного подсчета, т. к. каждое изменение бита влияет на последующее изменение других, т. к. может инвертировать их. Поэтому меняем знак для корректного подсчета.

4. Экспериментальная часть

Подсчёт по памяти *O(1)* т. к. все структуры имеют постоянный заранее объявленный размер. Вектор dp — 31*4=124Байта, переменные 4+4=8Байт, переменная в цикле — 4 байта. Итого 136 байт.

Подсчёт асимптотики *O(1)* т. к. все операции выполняются за фиксированное количество времени, дан фиксированный массив.

5. Заключение

Динамическое программирование было выбрано для решения этой задачи потому, что оно позволяет эффективно разбивать сложную задачу на более мелкие подзадачи и хранить промежуточные результаты, избегая повторных вычислений. Это важно там, где подзадачи перекрываются и их результаты могут быть использованы многократно. В данном случае, использование динамического программирования позволило минимизировать количество операций и обеспечить оптимальное решение задачи.

Другие подходы были бы неэффективны из-за нужды в многократном повторном вычислении одних и тех же операций, что привело бы к большому росту времени и памяти.

6. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла main.cpp

```cpp
#include <vector>

using std::vector;

class Solution {

public:
    int minimumOneBitOperations(int n) {
        if(n==0){
            return 0;
            }
        // 31 бит потому что этого хватает для любых int переменных
        vector<int> dp(31,0);
        dp[0]=1;
        for(int i = 1;i < 31; i++){
            dp[i]=dp[i-1]*2 + 1;
        }
        int steps=0;
        int sign=1;
        for(int i = 30;i >= 0; i--){
            if(n & (1 << i)){
                steps += dp[i] * sign;
                sign = 0 - sign;
            }

        }
        return steps;
    }
};
```