Лабораторная работа № 7 по курсу дискретного анализа: строковые алгоритмы

Выполнил студент группы М80-308Б-22 МАИ *Цирулев Николай*.

Условие

Вариант №4:
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.

- Формат ввода:

  В первой строке строке задано $2 \leq n \leq 10^7$

- Формат вывода:

  Выведите на первой строке искомую наименьшую стоимость. Во второй строке должна содержаться последовательность операций. Если было произведено деление на 2 или на 3, выведите */2* (или */3*). Если же было вычитание, выведите *-1*. Все операции выводите разделяя пробелом.

Метод решения

Динамическое программирование позволяет разбить сложную задачу оптимизации на более простые подзадачи.
Данную лабораторную работу оптимальней всего можно решить используя восходящее динамическое программирование. Идея решения состоит в том, чтобы сначала посчитать минимальную стоимость для всех чисел от 2 до n и в дальнейшем использовать эти значения для вычисления ответа.

Описание программы

Создадим вектор *std::vector<long long> c()*, в котором будем хранить минимальную стоимость для преобразования i в *1*. Создадим вектор *std::vector<long long> steps()* для хранения последней использованной операции (вычитание, деление на 2 или 3) для получения этой минимальной стоимости. Считаем значения массивов в первом цикле. После заполнения массивов, проводим обратный проход по массиву *steps*, начиная с n и двигаясь к 1, применяя к i одну из трех операций в зависимости от значения массива. В этом процессе формируется последовательность операций, примененных для достижения минимальной стоимости.

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

using namespace std;
using ll = long long;

int main() {
    ll n;
    cin >> n;
    vector<ll> steps(n + 1, 0);
    vector<ll> c(n + 1, numeric_limits<ll>::max());
    c[1] = 0;

    for (ll i = 2; i <= n; ++i) {
        if (c[i - 1] + i < c[i]) {
            c[i] = c[i - 1] + i;
            steps[i] = -1;
        }
        if (i % 2 == 0 && c[i / 2] + i < c[i]) {
            c[i] = c[i / 2] + i;
            steps[i] = 2;
        }
        if (i % 3 == 0 && c[i / 3] + i < c[i]) {
            c[i] = c[i / 3] + i;
            steps[i] = 3;
        }
    }
    vector<string> res;
    ll i = n;
    while (i > 1) {
        if (steps[i] == -1) {
            res.push_back("-1");
```

```
            i -= 1;
        } else if (steps[i] == 2) {
            res.push_back("/2");
            i /= 2;
        } else if (steps[i] == 3) {
            res.push_back("/3");
            i /= 3;
        }
    }
    cout << c[n] << endl;
    for (ll i = 0; i < res.size(); i++) {
        cout << res[i] << (i == res.size() - 1 ? "" : "␣");
    }
    cout << endl;
    return 0;
}
```

Дневник отладки

После отправки решения в чекер не было обнаружено ошибок.

Тест производительности

Асимптотика написанного алгоритма - *O(n)*.

Для лучшей наглядности приведём таблицу, в которой время работы алгоритма сопоставляется с вводимыми данными.

n	Время работы алгоритма, мс
1000	68
10000	544
100000	5290
1000000	53509
10000000	545919
100000000	5292997

Как показано в таблице, время выполнения алгоритма демонстрирует практически линейную зависимость от n. Это соответствует ожидаемой сложности нашего алгоритма. Ниже приведена программа benchmark.cpp, использовавшаяся для определения времени работы функций:

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

```cpp
#include <chrono>
#include <random>

using namespace std;
using ll = long long;

void Task(ll n) {
    vector<ll> steps(n + 1, 0);
    vector<ll> c(n + 1, numeric_limits<ll>::max());
    c[1] = 0;

    for (ll i = 2; i <= n; ++i) {
        if (c[i - 1] + i < c[i]) {
            c[i] = c[i - 1] + i;
            steps[i] = -1;
        }
        if (i % 2 == 0 && c[i / 2] + i < c[i]) {
            c[i] = c[i / 2] + i;
            steps[i] = 2;
        }
        if (i % 3 == 0 && c[i / 3] + i < c[i]) {
            c[i] = c[i / 3] + i;
            steps[i] = 3;
        }
    }
    vector<string> res;
    ll i = n;
    while (i > 1) {
        if (steps[i] == -1) {
            res.push_back("-1");
            i -= 1;
        } else if (steps[i] == 2) {
            res.push_back("/2");
            i /= 2;
        } else if (steps[i] == 3) {
            res.push_back("/3");
            i /= 3;
        }
    }
}

int main() {
```

```
    for (ll n = 10e2; n < 10e8; n *= 10) {
        std::cout << n << " & ";
        auto start = std::chrono::high_resolution_clock::now();
        Task(n);
        auto finish = std::chrono::high_resolution_clock::now();
        auto duration = std::chrono::duration_cast<std::chrono::microsecond
        std::cout << duration.count() << " \\" << " \\ ";
        std::cout << '\n';
    }
    return 0;
}
```

Выводы

В ходе выполнения данной работы были глубоко изучены принципы динамического программирования. Также в результате выполнения лабораторной работы был успешно реализован алгоритм на основе динамического программирования для минимизации стоимости преобразования числа n в единицу с использованием трех операций. Алгоритм продемонстрировал высокую эффективность с асимптотикой $O(n)$, обеспечивая оптимальное решение даже для больших входных данных. Практическое применение данного подхода подтверждает его значимость в задачах оптимизации и ресурсного управления.