ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

Отчёт по лабораторной работе № 6

«Динамическое программирование. Задача 1406 “Stone game III”»

Выполнил работу

Морозов Дмитрий

Академическая группа №J3110

Принято

Ментор, Вершинин Владислав

Санкт-Петербург

2024

1. Введение

Цель работы: решить задачу, используя динамическое программирование

Задачи:

­­­­­­­­­­— Провести анализ задачи

—Реализовать алгоритм, основанный на динамическом программировании

— Проанализировать затрачиваемую память и время работы программ

1. Реализация
   1. Анализ задачи, почему именно динамическое программирование

Условие задачи приведено на изображении №1. Согласно нему, нам надо определить, кто победит в игре. Чтобы узнать результат, мы должны знать каждый ход, который был сделан. Также мы знаем, что оба игрока совершают только лучшие для себя ходы. Так как при расчете хода каждый анализирует последующие возможные ходы, то мы можем сохранять все сделанные ходы, начиная с самого последнего. Таким образом нам не придется просчитывать всю цепочку ходов каждый раз заново, мы уже будем иметь нужные значение. Поэтому данную задачу требуется решать с помощью динамического программирования, так как мы накапливаем массив результатов, позволяющий быстро определять новый ход.

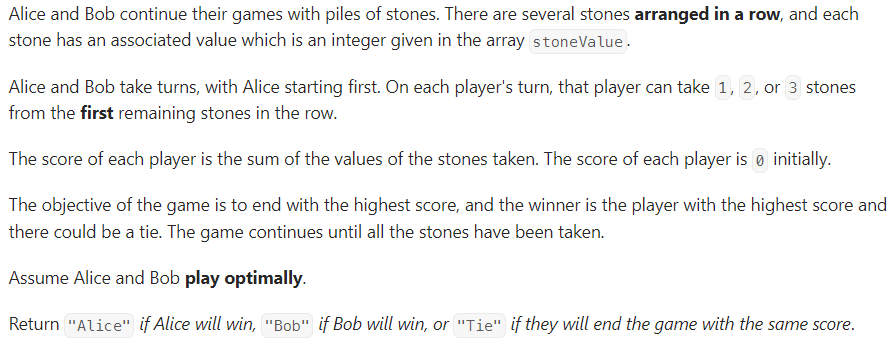


Рисунок №1 – условие задачи

* 1. Написание программы
     1. Реализация solve

На данном этапе была реализована функция solve, которая определяет максимальный счёт, который может быть достигнут из i-ой позиции. В stoneValue хранится вес каждого камня, cur – текущая позиция, откуда делается ход, all\_res – максимальное количество очков, которое можно получить из каждой i-ой позиции, n – количество камней. Сначала мы проверяем на каком элементе находимся. Как только дойдем до последнего, функция закончит свою работу. В противном случае проверяется, встречался ли нам этот ход ранее. Если нет, то мы проходимся по всем возможным ходам из этой позиции и выбираем лучший. Этот результат перезаписывается в all\_res, чтобы мы могли дальше его использовать. Реализация solve представлена на изображении №2.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок №2 – реализация функции solve

* + 1. Реализация stoneGameIII

На данном этапе был реализована функция stoneGameIII. Она определяет победителя для заданного набора камней. Для этого вызывается функция solve, которая в итоге вернёт максимальное количество очков, которое можно получить. Это и является результатом игры. Дальше он сравнивается с 0, чтобы определить победителя. Реализация stoneGameIII представлена на изображении №3.

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Рисунок №3 – Реализация stoneGameIII

* 1. Экспериментальная часть работы

На данном этапе производилась оценка затрачиваемой памяти и асимптотики. Также была осуществлена проверка встроенными тестами

* 1. Анализ проделанной работы

На данном этапе был выполнен анализ проделанной работы, удалось ли достичь необходимую цель и выполнить задачи.

1. Экспериментальная часть

На данном этапе производилась оценка затрачиваемой памяти и асимптотики. Также была осуществлена проверка встроенными тестами

Подсчёт по памяти:

Векторы stoneValue и res имеют размер n и содержат в себе целые числа int, поэтому на каждый вектор выделяется n\*4 байт. Кроме этого, выделяется 4 байта под переменную sum, которая пересоздаётся m раз, где m ≤ n. Итого общее количество используемой памяти: n\*4 + n\*4 + m\*4 = n\*8 + 4\*m байт.

Подсчёт асимптотики:

Функция stoneGameIII работает за О(1), так как не использует циклов или других затратных конструкций. Функция solve сначала проверяет элемент на нахождение в векторе, а также проверяется, закончилась ли игра. Это делается за О(1). Если это ложно, то запускается цикл, который перебирает до трёх возможных результатов. Так как все наши результаты хранятся в all\_res, то программе не надо заново просчитывать все варианты, поэтому сложность составляет О(n). Итоговая сложность всей программы составляет O(1) + O(1) ) + O(n) = O(n)

Проверка тестами:  
Написанная программа успешно прошла все встроенные тесты. Результаты представлены на изображении №4.

Изображение выглядит как текст, программное обеспечение, Значок на компьютере, веб-страница

Автоматически созданное описание

Рисунок №4 – Доказательство решения задачи

1. Заключение

В ходе лабораторной работы мною была решена задача с использованием динамического программирования. Данный подход был здесь уместен, это подтверждают успешные результаты работы. Анализируя пространственную и временную сложность, можно заметить, что динамическое программирование позволяет снизить расходы времени, но из-за этого появляются дополнительные расходы памяти.

ПРИЛОЖЕНИЕ А

Листинг кода файла lab\_6.cpp

#include <iostream>

#include <vector>

#include <string>

class Solution {

public:

    int solve(const std::vector<int> &stoneValue, const int& cur, std::vector<int> &all\_res, const int& n)

{

    if (cur == n)

    {

        return 0;

    }

    if (all\_res[cur] > -1e9)

    {

        return all\_res[cur];

    }

    int sum = 0;

    for (int i = cur; (i < cur + 3) and (i < n); i++)

    {

        sum += stoneValue[i];

        all\_res[cur] = std::max(all\_res[cur], sum - solve(stoneValue, i + 1, all\_res, n));

    }

    return all\_res[cur];

}

std::string stoneGameIII(const vector<int> &stoneValue)

{

    int n = stoneValue.size();

    std::vector<int> all\_res(n, -1e9);

    int score = solve(stoneValue, 0, all\_res, n);

    if (score > 0) {

        return "Alice";

    }

    else if (score < 0) {

        return "Bob";

    }

    else {

        return "Tie";

    }

}

};