

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

Отчёт по лабораторной работе № 7
«Жадные алгоритмы»

Выполнил работу
Ширкунова Мария
Академическая группа №J3114
Принято
Дунаев Максим Владимирович

Санкт-Петербург

2024

Содержание отчета

1. Введение.....	3
2. Реализация	4
3. Экспериментальная часть	6
4. Заключение	9
5. Приложения	10

1. Введение

Цель работы: решение задачи уровня hard на платформе leetcode, используя метод решения жадный алгоритм.

В рамках работы необходимо решить задачу 2448. Minimum Cost to Make Array Equal.

Задачи:

- Реализовать алгоритм.
- Протестировать алгоритм на платформе.
- Оценить сложность алгоритма.
- Оценить использование дополнительной памяти.
- Проанализировать, почему в решении необходим жадный алгоритм.

2. Реализация

Инициализируем переменные: `minimumCost` инициализируется значением `LLONG_MAX`, чтобы хранить минимальную стоимость, `prefixSum` — массив, который будет хранить префиксные суммы стоимостей, `pairs` — вектор пар, который будет хранить значения из `nums` и соответствующие им стоимости из `cost`.

В цикле `for` заполняется вектор `pairs`, где каждый элемент представляет собой пару (`nums[i]`, `cost[i]`). Пары сортируются по значениям из `nums`. Вектор `prefixSum` заполняется, где каждый элемент представляет собой сумму стоимостей до текущего индекса.

Переменная `rightCost` инициализируется как стоимость преобразования всех элементов к первому элементу (после сортировки). Это делается в цикле, где для каждого элемента (начиная со второго) вычисляется стоимость операций, необходимых для приведения его к значению первого элемента. Минимальная стоимость обновляется с учетом текущих значений левой и правой стоимости.

В следующем цикле происходит итерация по всем элементам (начиная со второго) для обновления левой и правой стоимости. `heightDifference` — это разница между текущим и предыдущим элементом. Правые и левые стоимости обновляются на основе этой разницы и префиксных сумм. Минимальная стоимость обновляется на каждом шаге. В конце функция возвращает минимальную стоимость, необходимую для приведения всех элементов массива к одному значению.

```
class Solution {
public:
    long long minCost(vector<int>& nums, vector<int>& cost) {

        long long minimumCost = LLONG_MAX;
        vector<long long> prefixSum;
        vector<pair<int, int>> pairs;

        for (int i = 0; i < cost.size(); i++) {
            pairs.push_back(make_pair(nums[i], cost[i]));
        }
    }
};
```

```

sort(pairs.begin(), pairs.end());

prefixSum.push_back(0);
for (int i = 0; i < pairs.size(); i++) {
    prefixSum.push_back(prefixSum[i] + pairs[i].second);
}

long long leftCost = 0, rightCost = 0;
for (int j = 1; j < cost.size(); j++) {
    rightCost += static_cast<long long>(pairs[j].first - pairs[0].first) *
static_cast<long long>(pairs[j].second);
}

minimumCost = min(minimumCost, leftCost + rightCost);

for (int i = 1; i < nums.size(); i++) {
    long long heightDifference = pairs[i].first - pairs[i - 1].first;
    rightCost -= heightDifference * static_cast<long long>(pairs[i].second);
    rightCost -= heightDifference * (prefixSum[prefixSum.size() - 1] -
prefixSum[i + 1]);
    leftCost += heightDifference * (prefixSum[i] - prefixSum[0]);
    minimumCost = min(minimumCost, leftCost + rightCost);
}

return minimumCost;
}
};

```

3. Экспериментальная часть

3.1. Подсчет по памяти.

```
class Solution {
public:
    long long minCost(vector<int>& nums, vector<int>& cost) {

        long long minimumCost = LLONG_MAX; // 8 байт
        vector<long long> prefixSum; // 4*N байт
        vector<pair<int, int>> pairs; // 2*4*N байт

        for (int i = 0; i < cost.size(); i++) { // 4 байт
            pairs.push_back(make_pair(nums[i], cost[i]));
        }
        sort(pairs.begin(), pairs.end());

        prefixSum.push_back(0);
        for (int i = 0; i < pairs.size(); i++) { // 4 байт
            prefixSum.push_back(prefixSum[i] + pairs[i].second);
        }

        long long leftCost = 0, rightCost = 0; // 2*8 байт
        for (int j = 1; j < cost.size(); j++) { // 4 байт
            rightCost += static_cast<long long>(pairs[j].first - pairs[0].first) *
static_cast<long long>(pairs[j].second);
        }

        minimumCost = min(minimumCost, leftCost + rightCost);

        for (int i = 1; i < nums.size(); i++) { // 4 байт
            long long heightDifference = pairs[i].first - pairs[i - 1].first; // 8 байт
            rightCost -= heightDifference * static_cast<long long>(pairs[i].second);
            rightCost -= heightDifference * (prefixSum[prefixSum.size() - 1] -
prefixSum[i + 1]);
            leftCost += heightDifference * (prefixSum[i] - prefixSum[0]);
            minimumCost = min(minimumCost, leftCost + rightCost);
        }

        return minimumCost;
        // Итого: 12N + 48 байт
    }
};
```

3.2. Подсчет асимптотики.

```
class Solution {
public:
    long long minCost(vector<int>& nums, vector<int>& cost) {
        // Пусть N=nums.size()==cost.size()

        long long minimumCost = LLONG_MAX;
        vector<long long> prefixSum;
        vector<pair<int, int>> pairs;

        for (int i = 0; i < cost.size(); i++) { // Цикл O(N)
            pairs.push_back(make_pair(nums[i], cost[i])); // O(1)
        }
        sort(pairs.begin(), pairs.end()); // Сортировка O(NlogN)

        prefixSum.push_back(0); // O(1)
        for (int i = 0; i < pairs.size(); i++) { // Цикл O(N)
            prefixSum.push_back(prefixSum[i] + pairs[i].second); // O(1)
        }

        return minimumCost;
    }
};
```

```

    }

    long long leftCost = 0, rightCost = 0;
    for (int j = 1; j < cost.size(); j++) { // Цикл O(N)
        rightCost += static_cast<long long>(pairs[j].first - pairs[0].first) *
static_cast<long long>(pairs[j].second); // O(1)
    }

    minimumCost = min(minimumCost, leftCost + rightCost); // O(1)

    for (int i = 1; i < nums.size(); i++) { // Цикл O(N)
        long long heightDifference = pairs[i].first - pairs[i - 1].first; // O(1)
        rightCost -= heightDifference * static_cast<long long>(pairs[i].second); //
O(1)
        rightCost -= heightDifference * (prefixSum[prefixSum.size() - 1] -
prefixSum[i + 1]); // O(1)
        leftCost += heightDifference * (prefixSum[i] - prefixSum[0]); // O(1)
        minimumCost = min(minimumCost, leftCost + rightCost); // O(1)
    }

    return minimumCost;
    // Итого:
    4*O(N)+O(NlogN)=O(4N)+O(NlogN)=O(N)+O(NlogN)=O(N+NlogN)=O(N(logN+1))=O(NlogN)
}
};

```

3.3. Прохождение тестов на leetcode.

The screenshot shows the LeetCode interface for the problem "Minimum Cost to Make Array Equal". The left sidebar displays the submission history, showing two "Accepted" submissions and one "Runtime Error". The main area shows the C++ code for the solution. The right sidebar shows the "Test Result" for Case 1, which is "Accepted" with a runtime of 0 ms. The input is `nums = [1, 3, 5, 2]` and `cost = [2, 3, 1, 14]`, resulting in an output of 8.

Minimum Cost to Make Array Equal - LeetCode

Accepted | Runtime: 8 ms | Beats: 80.60% | Memory: 48.71 MB | Beats: 15.60%

```

class Solution {
public:
    long long minCost(vector<int>& nums, vector<int>& cost) {
        long long minimumCost = LLONG_MAX;
        vector<long long> prefixSum;
        vector<pair<int, int>> pairs;

        for (int i = 0; i < cost.size(); i++) {
            pairs.push_back(make_pair(nums[i], cost[i]));
        }
        sort(pairs.begin(), pairs.end());
        // prefixSum.push back(0);
    }
};
  
```

Code | C++

```

class Solution {
public:
    long long minCost(vector<int>& nums, vector<int>& cost) {
        long long minimumCost = LLONG_MAX;
        vector<long long> prefixSum;
        vector<pair<int, int>> pairs;

        for (int i = 0; i < cost.size(); i++) {
            pairs.push_back(make_pair(nums[i], cost[i]));
        }
        sort(pairs.begin(), pairs.end());
        // prefixSum.push back(0);
    }
};
  
```

Testcase | Test Result

Accepted Runtime: 0 ms

- Case 1
- Case 2

Input

nums = [1,3,5,2]

cost = [2,3,1,14]

Output

8

4. Заключение

В ходе выполнения лабораторной работы был реализован алгоритм с использованием жадного подхода для решения задачи на leetcode о нахождении минимальной общей стоимости входного массива, при которой все его элементы станут одинаковыми.

Задача требует жадного подхода, так как чтобы эффективно находить оптимальное решение, нужно рассматривать каждое значение в отсортированном массиве как потенциальное целевое значение и выбрать то, которое дает наименьшую общую стоимость. Я использую локальные оптимальные решения (что и характерно для жадного алгоритма): алгоритм обновляет значения `left` и `right`, основываясь на текущем элементе массива. На каждом шаге он принимает решение о том, как минимизировать стоимость, добавляя или вычитая стоимость, связанную с текущим элементом.

Цель работы была достигнута путем тестирования алгоритма на платформе. Полученные результаты подтвердили теоретические оценки сложности алгоритма и эффективность решения задач жадным подходом.

5. Приложения

ПРИЛОЖЕНИЕ А

Листинг кода файла lab-7.cpp

```
class Solution {
public:
    long long minCost(vector<int>& nums, vector<int>& cost) {

        long long minimumCost = LLONG_MAX;
        vector<long long> prefixSum;
        vector<pair<int, int>> pairs;

        for (int i = 0; i < cost.size(); i++) {
            pairs.push_back(make_pair(nums[i], cost[i]));
        }
        sort(pairs.begin(), pairs.end());

        prefixSum.push_back(0);
        for (int i = 0; i < pairs.size(); i++) {
            prefixSum.push_back(prefixSum[i] + pairs[i].second);
        }

        long long leftCost = 0, rightCost = 0;
        for (int j = 1; j < cost.size(); j++) {
            rightCost += static_cast<long long>(pairs[j].first - pairs[0].first) *
static_cast<long long>(pairs[j].second);
        }

        minimumCost = min(minimumCost, leftCost + rightCost);

        for (int i = 1; i < nums.size(); i++) {
            long long heightDifference = pairs[i].first - pairs[i - 1].first;
            rightCost -= heightDifference * static_cast<long long>(pairs[i].second);
            rightCost -= heightDifference * (prefixSum[prefixSum.size() - 1] -
prefixSum[i + 1]);
            leftCost += heightDifference * (prefixSum[i] - prefixSum[0]);
            minimumCost = min(minimumCost, leftCost + rightCost);
        }

        return minimumCost;
    }
};
```