Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3233
Богатов Александр Сергеевич

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2022

Задача E: Коровы и стойла.

```
#include <iostream>

using namespace std;

int main()
{
    int n, k;

    cin >> n >> k;

    int places[n];

    for (int i = 0; i < n; i++)
        cin >> places[i];

    int closest = 0;

    int furthest = places[n-1] - places[0];

    while (furthest > closest) {
        int middle = (closest + furthest) / 2;
        int current_cow = places[0];
        int counter = 1;
        for (int i = 0; i < n; i++) {
            if (places[i] - current_cow > middle) {
                current_cow = places[i];
                counter++;
            }
        }
        if (counter >= k) {
            closest = middle + 1;
        } else furthest = middle;
    }
    cout << closest;
    return 0;
}
```

Пояснение к примененному алгоритму:

Использую метод бинарного поиска: в качестве левой границы берем 0, в качестве правой можем взять расстояние между первой и последней коровой, так мы покрываем весь отрезок потенциальных ответов. Далее выполняем поиск, пока границы не сойдутся: заведем переменную, в которой храним последнюю расселенную корову, с которой будем вести сравнение и заведем счетчик расселенных коров. В цикле для всех стойл смотрим на расстояние между стойлом и последней расселенной коровой – если оно превосходит текущую середину отрезка, то мы можем расселить корову не нарушив условие. По выходу с цикла смотрим, сколько мы расселили коров – если >= k, значит можно сдвинуть левую границу, иначе сдвигаем правую границу, так ищем ответ дальше деля отрезок пополам.

Задача F: Число

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

```
using namespace std;

int main()
{
    string current;

    vector<string> number;

    while (cin >> current) {
        if (number.empty())
            number.push_back(current);
        else {
            for (int i = number.size() - 1; i >= 0; i--) {
                if (number[i] + current > current + number[i]) {
                    number.insert(number.begin() + i + 1, current);
                    break;
                }
                if (i == 0) {
                    number.insert(number.begin(), current);
                }
            }
        }
    }

    for (int i = 0; i < number.size(); i++) {
        cout << number[i];
    }
    return 0;
}
```

Пояснение к примененному алгоритму:

Храним введенные значения в векторе, если вектор не пустой, для всех значений в векторе (идя с вершины) проведем сравнение конкатенации строк элемента вектора с текущим элементов и наоборот. В случае, если прибавление текущего элемента за элементом вектора больше, вставляем текущий элемент в вектор на позицию над текущим элементом. Если подобной позиции для текущего числа не найдено, вставляем его в самое начало вектора. Так мы добиваемся максимального числа при сложении всех строк в векторе начиная с его начала.

Задача G: Кошмар в замке

```
#include <iostream>
#include <map>

using namespace std;

int main()
{
    string current;

    cin >> current;

    int weight[26];

    char alphabet[26];
```

```
char base = 'a';

int counter[26];

string result = "";

map<char, int> postSortIdxs;

for (int i = 0; i < 26; i++) {
    cin >> weight[i];
    alphabet[i] = base + i;
    counter[i] = 0;
}

for (int i = 1; i < 26; i++) {
        int value = weight[i];
        int alpha_value = alphabet[i];
        int j = i;
        while (j > 0 && weight[j - 1] < value) {
            weight[j] = weight[j - 1];
            alphabet[j] = alphabet[j-1];
            j--;
        }
        weight[j] = value;
        alphabet[j] = alpha_value;
}

for (int i = 0; i < 26; i++) {
    postSortIdxs[alphabet[i]] = i;
}

for (int i = 0; i < current.size(); i++) {
    counter[postSortIdxs[current[i]]]++;
}

for (int i = 0; i < 26; i++) {
    if (counter[i] >= 2) {
       result += alphabet[i];
    }
}

for (int i = 0; i < 26; i++) {
    if (counter[i] == 1)
        result += alphabet[i];
}

for (int i = 0; i < 26; i++) {
    while (counter[i] > 2) {
            result += alphabet[i];
            counter[i]--;
    }
}

for (int i = 25; i >= 0; i--) {
    if (counter[i] >= 2)
        result += alphabet[i];
}
```

```
    cout << result << endl;

    return 0;
}
```

Пояснение к примененному алгоритму:

Заведем массив со всеми весами букв, массив с буквами для соответствия и массив счетчиков вхождений буквы в строку. Полученные веса мы отсортируем алгоритмом обратной сортировки вставкой: запоминаем текущее значение во временной переменной, далее идем по элементам слева пока они меньше запомненного элемента, вставляем элемент, когда находим больший элемент. Параллельно с сортировкой массива весов, сортируем на основе сравнений связанных с первым массивом массив букв английского алфавита, чтобы сохранить соответствие между буквами и их весами. Далее заведем мапу хранящую индексы букв алфавита после сортировки. Пойдем по символам строки: будем увеличивать счетчик вхождений соответствующей буквы, когда мы ее нашли – индекс увеличенного счетчика = индексу буквы в мапе.

В первую очередь к результату прибавляем самые тяжелые буквы, входящие в строку 2 или более раз. Далее прибавляем буквы с 1 вхождением, ведь они веса не имеют. Далее прибавляем буквы с > 2 вхождениями начиная с самой тяжелой, параллельно уменьшая счетчик вхождений пока он не опустится до двух. В конце прибавляем буквы с >2 вхождениями начиная с самых легких – так самые тяжелые окажутся в конце строки и вес строки будет больше.

Задача H: Магазин

```
#include <iostream>

using namespace std;

int main()
{
    int n, k;
    cin >> n >> k;
    int bucket[n];

    for (int i = 0; i < n; i++) {
        cin >> bucket[i];
    }

    int sum = 0;

    int discount = n/k;

    for (int i = 1; i < n; i++) {
            int value = bucket[i];
            int j = i;

            while (j > 0 && bucket[j - 1] < value) {
                bucket[j] = bucket[j - 1];
                j--;
            }

            bucket[j] = value;
```

```
    }

    for (int i = 0; i < n; i++) {
        if ((i+1)%k != 0)
            sum += bucket[i];
    }

    cout << sum << endl;

    return 0;
}
```

Пояснение к примененному алгоритму:

Сортировкой вставкой отсортируем цены купленных товаров в убывающем порядке, теперь каждый k-ый элемент мы можем выкинуть из чека и при этом максимизировать скидку.