Федеральное государственное автономное
образовательное учреждение высшего образования
«Научно-образовательная корпорация ИТМО»

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Программная инженерия

Отчёт по лабораторной работе №3

По дисциплине «Алгоритмы и структуры данных» (4 семестр)
Второй блок задач

Студент:
Дениченко Александр P3212
Преподаватели:
Косяков М.С.
Тараканов Д.С.

Санкт-Петербург
2024 г.

1 Задача №E «Коровы в стойла»

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

using namespace std;

int main()
{
    int N, K;
    cin >> N >> K;

    vector<int> stalls(N);
    for (int i = 0; i < N; ++i)
    {
        cin >> stalls[i];
    }

    int left = 1;
    int right = stalls.back() - stalls.front();
    int result;

    for (;;)
    {
        if (left > right)
        {
            cout << result << endl;
            return 0;
        }
        int mid = left + (right - left) / 2;
        int sit = 1;
        int last_sit = 0;
        for (int i = 1; i < stalls.size(); ++i)
        {
            if (stalls[i] - stalls[last_sit] >= mid)
            {
                sit++;
                last_sit = i;
                if (sit == K)
                {
                    result = mid;
                    left = mid + 1;
                    break;
                };
            }
        }
        if (sit != K)
        {
            right = mid - 1;
        }
    }
}
```

Запускается бесконечный цикл, в котором выполняется бинарный поиск оптимального расстояния между коровами. На каждой итерации бинарного поиска вычисляется значение mid как середина текущего диапазона left и right. Затем происходит итерация по стойлам, чтобы определить, сколько коров можно разместить с заданным расстоянием mid между

ними. Если количество размещенных коров равно K, то сохраняется текущее значение mid как потенциальный результат, и диапазон поиска сужается до правой половины (left = mid + 1). В противном случае диапазон поиска сужается до левой половины (right = mid - 1). Этот алгоритм гарантирует, что минимальное расстояние между коровами будет максимально возможным, и решение будет найдено за логарифмическое время от размера интервала между самым маленьким и самым большим расстояниями между стойлами.

2 Задача №F «Число»

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

using namespace std;

int partition(vector<string> &arr, int low, int high)
{
    string op = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++)
    {
        string first_pair = arr[j] + op;
        string second_pair = op + arr[j];
        if (first_pair > second_pair)
        {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}

void qSort(vector<string> &arr, int low, int high)
{
    if (low < high)
    {
        int pi = partition(arr, low, high);

        qSort(arr, low, pi - 1);
        qSort(arr, pi + 1, high);
    }
}

int main()
{
    vector<string> particles;
    string inp;
    while (cin >> inp)
    {
        particles.push_back(inp);
    }
    // particles = {"10", "07", "8", "9", "0001", "05"};
    qSort(particles, 0, particles.size() - 1);

    for (string x : particles)
    {
        cout << x;
```

```
        }
        cout << endl;
        return 0;
    }
```

Этот код решает задачу определения максимального числа, которое могло быть написано на полоске бумаги до ее разрезания. Он использует алгоритм быстрой сортировки (quicksort) для сортировки частей полоски бумаги. Функция partition разделяет массив частей полоски бумаги на две части относительно опорного элемента, который выбирается как последний элемент массива. Затем функция qSort рекурсивно сортирует обе половины массива путем вызова partition. В функции main считываются части полоски бумаги и сохраняются в вектор particles. Затем вызывается функция qSort для сортировки частей. В результате сортировки части объединяются вместе, чтобы образовать наибольшее число. Алгоритм быстрой сортировки имеет среднюю временную сложность O(n log n), но в худшем случае, когда опорный элемент каждый раз выбирается как наименьший или наибольший элемент массива, сложность может составить O(n^2).

3 Задача №G "Кошмар в замке"

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

using namespace std;

struct Br
{
    int cost;
    int count;
};

int findMaxCostBrIndex(const vector<Br> &korob_p)
{
    int maxCost = -1;
    int maxCostIndex = 0;

    for (size_t i = 0; i < korob_p.size(); ++i)
    {
        if (korob_p[i].count > 1 && korob_p[i].cost > maxCost)
        {
            maxCost = korob_p[i].cost;
            maxCostIndex = i;
        }
    }

    return maxCostIndex;
}

int main()
{
    string stroka, costs_p;
    cin >> stroka;

    vector<Br> korob_p(26); //'a'+i

    for (int i = 0; i < 26; i++)
    {
        cin >> korob_p[i].cost;
    }
```

```
    for (int i = 0; i < stroka.size(); i++)
    {
        korob_p[stroka[i] - 'a'].count++;
    }

    int size_ps = stroka.size();
    string left_pochka;
    string center_pochka;

    while (size_ps > 0)
    {
        size_ps--;
        int max_l_p = findMaxCostBrIndex(korob_p);
        while (korob_p[max_l_p].count > 1)
        {
            left_pochka += 'a' + max_l_p;
            korob_p[max_l_p].count -= 2;
            while (korob_p[max_l_p].count > 0)
            {
                center_pochka += 'a' + max_l_p;
                korob_p[max_l_p].count--;
            }
        }
        if (korob_p[max_l_p].count == 1)
        {
            center_pochka += 'a' + max_l_p;
            korob_p[max_l_p].count--;
        }
    }

    for (int i = 0; i < 26; i++)
    {
        if (korob_p[i].count == 1)
        {
            center_pochka += 'a' + i;
            korob_p[i].count--;
        }
    }

    cout << left_pochka << center_pochka;
    for (int i = left_pochka.size() - 1; i >= 0; i--)
    {
        cout << left_pochka[i];
    }

    return 0;
}
```

Прощу не обращать внимание на спицифичные неймминги. На данный момент самая сложная задача, так как получилось её решить после 10-15 часов плотного дебага, но всё оказалось куда проще, не совсем очевидно из условия, что каждая буква может использоваться 1 раз для составления пары. (для строки aaaabbbb ответом будет: abXXXXba, а не aabbbbaa (a>b)) Метод, описанный в решении уже точно не получится сломать, так как происходит полный контроль и просчёт букв в строке. Создается вектор korob p, в котором каждая буква алфавита представлена структурой Br, содержащей стоимость и количество встреченных букв в строке. Далее, для каждой буквы из входной строки увеличивается соответствующее значение в векторе korob p. Затем выполняется процесс максимизации веса строки: На каждом шаге выбирается буква с максимальной стоимостью, у которой количество больше 1. Для выбранной буквы создается левая и

центральная части строки, которые формируются из повторяющихся экземпляров этой буквы. Если у выбранной буквы остается одна копия, она добавляется в центральную часть строки. После формирования центральной части строки все оставшиеся буквы с количеством 1 добавляются в конец центральной части. Наконец, левая часть строки добавляется перед центральной, а затем обе части выводятся. Таким образом, программа создает строку с максимально возможным весом, учитывая веса букв и расстояния между повторяющимися буквами. Кстати, у контеста нет интересной проверки на подобное:
ggggdgggpppqweqw
26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26 26
верный ответ должен всё равно подразумевать все возможные пары по краям.
Общая алгоритмическая сложность кода примерно составляет O(n^2) (из-за вложенных циклов, но имеется хорошая локальность данных)

4 Задача №H "Магазин"

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

using namespace std;

int main()
{

    int N, K;
    cin >> N >> K;
    int sum = 0;
    vector<int> products(N);
    for (int i = 0; i < N; ++i)
    {
        cin >> products[i];
        sum += products[i];
    }

    sort(products.begin(), products.end(), greater<int>());
    for (int i = K - 1; i < N; i += K)
    {
        sum -= products[i];
    }
    cout << sum;
    return 0;
}
```

Сначала считываются два числа N и K, где N - количество товаров, которые Билл хочет купить, а K - каждый K-й товар бесплатно. Затем считывается цена каждого товара и сохраняется в векторе products. Вычисляется общая сумма всех товаров и сохраняется в переменной sum. Товары сортируются по убыванию цены с помощью функции sort. Затем происходит итерация по товарам с шагом K-1 (товары, которые будут бесплатными), и из общей суммы вычитается цена каждого такого товара. Наконец, выводится минимальная сумма, которую нужно заплатить Биллу. Главное сортировать в верном порядке. Мы полностью предусамтриваем все случаи, поэтому код можно считать верным. Алгоритм имеет временную сложность O(N log N).