Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3216

Брагин Р.А.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2025

Задача E. Коровы в стойле.

```
#include <iostream>


using namespace std;
#include <vector>


bool proverkaRastanovki(vector<int>& stalls, int k, int dist) {
  int cnt = 1;
  int curr_cow = stalls[0];
  for (size_t i = 1; i < stalls.size(); i++) {
    if (-curr_cow + stalls[i] >= dist) {
      curr_cow = stalls[i];
      cnt++;
    }
    if (cnt == k) {
      return true;
    }
  }
  return false;
}


int binSearch(vector<int>& stalls, int k) {
  int left = 0;
  int right = stalls[stalls.size() - 1] - stalls[0] + 1;
  while (left + 1 < right) {
    int mid = (left + right) / 2;
    if (proverkaRastanovki(stalls, k, mid)) {
      left = mid;
    } else {
```

```
      right = mid;
    }
  }
  return left;
}

int main() {
  int n, k;
  cin >> n >> k;
  vector<int> stalls(n);
  for (int& x : stalls)
    cin >> x;
  int answer = binSearch(stalls, k);
  cout << answer << endl;
  return 0;
}
```

Описание: Алгоритм ищет такое максимальное расстояние, чтобы k коров можно было расставить по n стойлам. Для этого сначала сортируем стойла и применяем бинарный поиск по расстоянию: пробуем поставить коров с промежутком mid и смотрим, получается ли. Если коровы встают, пробуем увеличить расстояние, иначе уменьшаем. В конце остаётся максимальное возможное расстояние, а сложность алгоритма —

O(n log n). В худшем случае потребление памяти — O(n)

Задача F. Число.

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

using namespace std;

bool comparator(const string& a, const string& b) {
  if (a[0] != b[0]) {
    return a[0] > b[0];
  }

  return a + b > b + a;
}

int main() {
  vector<string> numbers;
  string line;
  while (cin >> line) {
    numbers.push_back(line);
  }

  sort(numbers.begin(), numbers.end(), comparator);

  string answer;
  for (string line : numbers) {
    answer += line;
  }
  cout << answer << endl;
  return 0;
}
```

Описание: Компаратор сравнивает строки сначала по первому символу, а если он совпадает, то проверяет, какая конкатенация даст больший результат. После сортировки строки объединяются в итоговую строку и выводятся. Сложность алгоритма O(n log n), а память O(n), так как храним вектор строк.

Задача G. Кошмар в замке.

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

using namespace std;

bool customCompare(const pair<char, int>& a, const pair<char, int>& b)
{
  return a.second > b.second;
}

unordered_map<char, int> calculateFrequency(const string& input) {
  unordered_map<char, int> frequency;
  for (char ch : input) {
    frequency[ch]++;
  }
  return frequency;
}

vector<pair<char, int>> readLetterValues() {
  vector<pair<char, int>> letterValues(26);
  for (int i = 0; i < 26; i++) {
    cin >> letterValues[i].second;
    letterValues[i].first = 'a' + i;
  }
  sort(letterValues.begin(), letterValues.end(), customCompare);
  return letterValues;
}

string constructResult(
    unordered_map<char, int>& frequency, const vector<pair<char,
int>>& letterValues
) {
  string leftPart, rightPart;
  for (const auto& entry : letterValues) {
    char ch = entry.first;
    if (frequency[ch] >= 2) {
      leftPart += ch;
```

```
      rightPart = ch + rightPart;
      frequency[ch] -= 2;
    }
  }

  string middlePart;
  for (const auto& entry : frequency) {
    if (entry.second > 0) {
      middlePart += string(entry.second, entry.first);
    }
  }

  return leftPart + middlePart + rightPart;
}

int main() {
  string input;
  cin >> input;

  unordered_map<char, int> frequency = calculateFrequency(input);
  vector<pair<char, int>> letterValues = readLetterValues();

  cout << constructResult(frequency, letterValues);

  return 0;
}
```

Описание: Сначала подсчитываем частоту каждого символа в строке. Затем считываем значения для каждой буквы и сортируем их по убыванию. После этого из символов с наибольшими значениями строим первую и последнюю части строки, добавляя символы попарно, если их частота больше или равна 2. Оставшиеся символы добавляются в середину строки. Сложность алгоритма — O(n) для сортировки и подсчета частоты, а по памяти — O(n), где n - длина входной строки.

Задача G. Число.

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

using namespace std;

int getCheapestSum(vector<int>& prices, int n, int k) {
  int total = 0;

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

  for (int i = 0; i < n; ++i) {
    if ((i + 1) % k != 0) {
      total += prices[prices.size() - i - 1];
    }
  }
  return total;
}

int main() {
  int n, k;
  cin >> n >> k;
  vector<int> prices(n);
  for (int& x : prices)
    cin >> x;

  cout << getCheapestSum(prices, n, k) << endl;
  return 0;
}
```

Описание: Сначала сортируем список цен по возрастанию. Затем, начиная с самой дорогой позиции, добавляем в сумму все цены, кроме каждых k самых дешевых товаров (они пропускаются). Сложность алгоритма — O(n log n) из-за сортировки, а по памяти — O(n), так как используется массив цен.