

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)

Студент:

Дениченко Александр Р3212

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург
2024 г.

1 Задача №А «Агроном-любитель»

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

int main()
{
    int n;
    std::cin >> n;
    std::vector<int> numbers(n);
    for (int i = 0; i < n; ++i)
    {
        std::cin >> numbers[i];
    }
    if (n == 1)
    {
        std::cout << "1 1";
        return 0;
    }
    if (n == 2)
    {
        std::cout << "1 2";
        return 0;
    }
    if (n == 3)
    {
        if (numbers[0] == numbers[1] && numbers[1] == numbers[2] && numbers[2] ==
            ↪ numbers[0])
        {
            std::cout << "1 2";
            return 0;
        }
        std::cout << "1 3";
        return 0;
    }
    std::vector<int> ans = {1, 2};
    std::vector<int> max = {1, 2};
    for (int i = 2; i < n; i++)
    {
        if (!(numbers[i - 2] == numbers[i - 1] && numbers[i - 1] == numbers[i] &&
            ↪ numbers[i - 2] == numbers[i]))
        {
            ans[1] = i + 1;
        }
        else
        {
            ans[0] = i;
            ans[1] = i + 1;
        }
        if ((max[1] - max[0] + 1) < (ans[1] - ans[0] + 1))
        {
            max = ans;
        }
    }
    std::cout << max[0] << " " << max[1];
    return 0;
}
```

```
}
```

Сначала проверяются базовые случаи: если на грядке всего один цветок, если два цветка и если три цветка. Создаются два вектора, которые будут хранить текущий ответ и максимальный найденный ответ соответственно. Проходя по всем цветкам на грядке, проверяется условие: если три цветка подряд одного вида не обнаружены, то увеличивается длина текущего участка. Если обнаружены три цветка подряд одного вида, то обновляется начало текущего участка. При каждом шаге обновляется максимальный найденный участок без трех цветков одного вида подряд. После завершения цикла выводится найденный наиболее длинный участок грядки без трех цветков одного вида подряд. Данный метод решения пришёл ко мне относительно быстро. Благодаря полному детальному проходу алгоритм может считаться точным. Данный проход перебирает все вариации ответов к задаче, поэтому является эффективным. Таким образом, алгоритм имеет линейную временную сложность в лучшем и худшем случаях.

2 Задача №В «Зоопарк Глеба»

```
#include <iostream>
#include <stack>

int main()
{
    std::string input_str;

    std::cin >> input_str;

    std::stack<int> upper_chars;
    std::stack<int> lower_chars;
    std::stack<char> current_str;
    int answer[input_str.length() / 2];
    int up_chars = 0;
    int lo_chars = 0;
    for (int i = 0; i < input_str.length(); i++)
    {
        if (isupper(input_str[i]))
        {
            up_chars++;
            upper_chars.push(up_chars);
            // std::cout << "add upper - " << input_str[i] << " number - " <<
            // up_chars << std::endl;
        }
        else
        {
            lo_chars++;
            lower_chars.push(lo_chars);
            // std::cout << "add lower - " << input_str[i] << " number - " <<
            // lo_chars << std::endl;
        }
        if (current_str.empty())
        {
            // std::cout << "empty" << std::endl;
            current_str.push(input_str[i]);
        }
        else
        {
            char last = current_str.top();
            if ((abs(last - input_str[i]) == 32))
            {
                answer[upper_chars.top() - 1] = lower_chars.top();
            }
        }
    }
}
```

```

        upper_chars.pop();
        lower_chars.pop();
        current_str.pop();
    }
    else
    {
        current_str.push(input_str[i]);
    }
}
if (!current_str.empty())
{
    std::cout << "Impossible" << std::endl;
    return 0;
}
std::cout << "Possible" << std::endl;
for (int i = 0; i < input_str.length() / 2; i++)
{
    std::cout << answer[i] << " ";
}
std::cout << std::endl;
return 0;
}

```

Идея разделения букв (животных и капканов) на стеки пришла ооочень не сразу, сначала был вообще другой подход, который предусматривал разделение букв в двумерный массив, где будет буква и его id. Так как я не смог увидеть ошибку в прошлом коде, то решил переписать всё с нуля и в итоге опять увидел ошибку, так как в самом начале нового кода я сделал `stack < char > upper_chars;` `stack < char > lower_chars;` где была опять логика сохранять буквы в их порядке, но далее был сделан вывод, что их нет смысла хранить. Я переделал код под хранение индексов в стеках, но забыл проверить пример с очень длинной строкой и `char` могут неверно отрабатывать, поэтому было решение сохранять индексы в `int`. Алгоритм работает за линейное время $O(n)$.

3 Задача №С «Конфигурационный файл»

```

#include <iostream>
#include <stack>
#include <unordered_map>

bool is_integer(const std::string &s)
{
    if (s.empty())
        return false;
    size_t start = (s[0] == '-' || s[0] == '+') ? 1 : 0;
    for (size_t i = start; i < s.length(); ++i)
    {
        if (!std::isdigit(s[i]))
        {
            return false;
        }
    }
    return true;
}

int main()
{
    std::unordered_map<std::string, std::stack<int>> map;
}

```

```

std::stack<std::string> new_var;
std::string line;

while (std::cin >> line)
{
    if (line.find('=') != std::string::npos)
    {
        std::string key = line.substr(0, line.find('='));
        std::string value = line.substr(line.find('=') + 1);
        // std::cout << "key: " << key << " value: " << value << std::endl;
        if (is_integer(value))
        {
            int int_value = std::stoi(value);
            map[key].push(int_value);
        }
        else
        {
            if (!map[value].empty())
            {
                map[key].push(map[value].top());
                std::cout << map[value].top() << std::endl;
            }
            else
            {
                map[key].push(0);
                std::cout << 0 << std::endl;
            }
        }
        new_var.push(key);
    }
    else
    {
        if (line == "{")
        {
            // std::cout << "new area" << std::endl;
            new_var.push("EOF");
        }
        else if (line == "}")
        {
            // std::cout << "end new area" << std::endl;
            while (new_var.top() != "EOF")
            {
                // std::cout << "clear " << new_var.top() << " : " <<
                ↪ map[new_var.top()].top() << std::endl;
                map[new_var.top()].pop();
                new_var.pop();
            }
            new_var.pop();
        }
        else
        {
            // {b=1000} ?? error 1: incorrect cin
            return 1;
        }
    }
}

```

```
    return 0;
}
```

Данная идея реализации мне пришла сразу, так как я ранее писал парсер на питоне для файлов json. Сначала была идея сделать *stack < unordered_map < string,int >> data;*, которая бы хранила в стеке актуальный регион, причём перед добавлением в стек я должен был копировать прошлую area, но этот подход требовал слишком много памяти, поэтому я пересмотрел решение. В итоге я пришел к алгоритму - класть в стек все переменные и ещё отдельно хранить изменения в текущей области. Алгоритм работает в среднем за линейное время $O(n)$.

4 Задача №D «Профессор Хаос»

```
#include <iostream>
#include <stack>
#include <unordered_map>

int main()
{
    int start;
    int new_vir;
    int delete_vir;
    int place_cont;
    int days;

    std::stack<int> curr;

    std::cin >> start >> new_vir >> delete_vir >> place_cont >> days;

    int test = start;

    for (int i = 0; i < days; i++)
    {
        start = start * new_vir;
        if (start <= delete_vir)
        {
            std::cout << 0;
            return 0;
        }
        else
        {
            start = start - delete_vir;
        }
        if (start >= place_cont)
        {
            start = place_cont;
        }
        if (test == start)
        {
            std::cout << start;
            return 0;
        }
    }
    std::cout << start;
    return 0;
}
```

Задача показалась тривиальной, формула, думаю, у всех одна и та же, но были пару трудностей, так как не было проверки на заикливание, но через несколько тестов понял где ошибка. Сложность алгоритма - линейная.

5 Задача №Е «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;
        }
    }
}
```

В задаче использовал идею двоичного поиска. 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.

6 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;
}

```

Сам алгоритм сразу возник в голове, код реализует алгоритм сортировки частиц и склеивает их, но решил вспомнить как адекватно прописать быструю сортировку. 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 $O(n^2)$.

7 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 специфичные нейминги. 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 `aabbbbba` ($a > b$)) Метод, описанный в решении уже точно не получится сломать, так как происходит полный контроль и подсчёт букв в строке. Кстати, у контекста нет интересной проверки на подобное:

ggggdgggpppqweqw

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)

8 Задача №Н "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;
}
```

Опять же задача сразу была написана напрямую из условия, подводных камней нет, главное сортировать в верном порядке. Алгоритм имеет временную сложность $O(N \log N)$.