Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по «Алгоритмам и структурам данных»

Timus

Выполнил:

Студент группы P3231

Савон Г.К.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2021

Задача 1322 «Шпион»

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

using namespace std;

int n;
string s;
vector<int> sort_str;

bool comp(int i, int j) { return s[j] > s[i]; }

int main() {
  cin >> n;
  cin >> s;
  int len = s.size();
  for (int i = 0; i < len; i++) {
    sort_str.push_back(i);
  }
  stable_sort(sort_str.begin(), sort_str.end(), comp);
  int now_num = sort_str[n - 1];
  for (int i = 0; i < len; i++) {
    cout << s[now_num];
    now_num = sort_str[now_num];
  }
  return 0;
}
```

Пояснение к примененному алгоритму:

Все рассуждения по этой задаче съела криса, так что рассказывать нечего…

На самом деле же мы знаем все буквы, которые используются в изначальной строке. А также знаем, что эти буквы, отсортированные по алфавиту – это первый столбик всей «матрицы» различных строк. А также понимаем, что если сместить все строчки на один вправо, то получатся также строки из матрицы. И они должны быть

отсортированы по алфавиту.

Отсортировали. Супер. А что мы знаем про отсортированные строки – знаем их последний столбец. Далее можем повторить все то же самое. Вот так вот до самого конца.

Но!!! Оказалось, что это слишком долго. Да и вообще зачем восстанавливать все строки, когда нужна только одна? И правда. Каждая буква из конца переходит на одно и то же место после того, как мы смещаем строки и сортируем их заново. Поэтому достаточно отсортировать их все всего один раз и запомнить, какая – куда пошла, и после – восстановить нужную нам по порядку строку по этим числам.

Сложность: поскольку при сортировке важно чтобы буквы сохраняли свой порядок (даже если сами буквы то одинаковые), то приходится использовать stable_sort вместо обычного sort. Так что сложность – $O(n\log^2(n))$

Задача 1604 «Страна Дураков»

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

using namespace std;

vector<pair<int, int> > speads;

bool comp(pair<int, int> p1, pair<int, int> p2) {
  if (p1.second <= p2.second) return 0;
  return 1;
}

void sorting() {
  int i = 1;
  pair<int, int> local_pair;
  while (i < speads.size() - 1 && (speads[i].second <= speads[i + 1].second)) {
    local_pair = speads[i];
    speads[i] = speads[i + 1];
    speads[i + 1] = local_pair;
    i++;
  }
  i = 0;
  while (i < speads.size() - 1 && (speads[i].second < speads[i + 1].second)) {
    local_pair = speads[i];
    speads[i] = speads[i + 1];
    speads[i + 1] = local_pair;
    i++;
  }
}
```

```
int main() {
  int k;
  cin >> k;
  int number_of_speads;

  if (k <= 1) {
    if (k != 0) {
      cin >> number_of_speads;
      for (int i = 0; i < number_of_speads; i++) cout << 1 << " ";
    }
  } else {
    for (int i = 0; i < k; i++) {
      cin >> number_of_speads;
      speads.push_back(make_pair(i + 1, number_of_speads));
    }
    sort(speads.begin(), speads.end(), comp);
    while (speads[0].second > 0 && speads[1].second > 0) {
      cout << speads[0].first << " " << speads[1].first << " ";
      speads[0].second--;
      speads[1].second--;
      sorting();
    }
    for (int i = 0; i < speads[0].second; i++) cout << speads[0].first << " ";
  }
  return 0;
}
```

Пояснение к примененному алгоритму:

Нужно чередовать знаки, поэтому первым безусловно надо поставить тот, которого больше всего изготовили.

На следующем шагу надо сделать то же самое, но не ставить знак, который был поставлен только что – то есть поставить второй по количеству. И так пока все не закончатся. НО! После выставления пары знаков нужно заново отсортировать все знаки по количеству, ведь двух первых уже стало меньше.

Сложность: первоначальная сортировка выполняется за O(nlog(n)), каждая последующая сортировка (я ее делаю просеиванием первых двух значений) – за n – в худшем случае. – и этих сортировок будет много – суммарное количество знаков.

O(nm), где n – количество видов знаков, m – сума их количества