Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2
по “Алгоритмам и структурам данных”
Базовые задачи

Выполнил:
Студент группы P3206
Сорокин А.Н.
Преподаватели:
Косяков М.С.
Тараканов Д.С.

Санкт-Петербург
2024

Задача №E “Коровы в стойла”

Решение за O(n log n).
Необходимо найти такое оптимальное расстояние между коровами в стойлах, чтобы оно было как можно больше.
Пусть мы имеем какое-то расстояние r . Тогда наши коровы должны находиться в стойлах на расстоянии $min(a_{i+1}-a_i)\geq r$. Если при данном r коров расставить не получилось, значит, при $r'>r$ их и подавно не получится расставить. Если же их получилось расставить, то при $r'<r$ их тем более получится расставить. Таким образом, можно сокращать область поиска оптимального значения r , отсекая целые промежутки на каждой итерации.
Чтобы ускорить поиск, на каждой итерации в качестве нового r рассматривается середина области. Таким образом, после каждой проверки на расстановку коров область поиска сокращается в половину. Данный алгоритм является алгоритмом бинарного поиска.
Отсюда такая ассимптотика: поиск значения за O(log n) и расстановка коров для каждого значения за O(n) дают O(n log n).

Код решения:

```
#include <cstddef>
#include <fstream>
#include <iostream>
#include <vector>

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);

  ifstream infile("input.txt");

  vector<long long> v;
  long long n, k, a;
  infile >> n >> k;
```

```
  while (infile >> a)
    v.push_back(a);

  long long l = 0, r = v.back() - v.front() + 1;
  while (l + 1 < r) {
    long long m = (l + r) / 2, last = 0, count = 1;
    for (long long i = 0; i < n; ++i) {
      if (v.at(i) - v.at(last) >= m) {
        count += 1;
        last = i;
      }
    }
    if (count < k) {
      r = m;
    } else {
      l = m;
    }
  }
  cout << l;
  return 0;
}
```

Задача №F “Число”

Решение за O(n log n).
Решение задачи заключается в написании компаратора для сортировки переданных чисел.
Поскольку задача требует составление максимально большого числа из переданных чисел, сравнение чисел a и b с разными длинами перестает быть тривиальной задачей. Например, очевидно, что $1<20$, однако для данной задачи $3>20$, поскольку $320>203$. Сравнивать числа одинаковых длин, тем не менее, гораздо проще: их можно сравнить “как обычно”.
Следовательно, для решения задачи любое сравнение необходимо свести к сравнению чисел одинаковой длины.
Подход к решению может быть следующим:

Пусть даны числа a и b : $\sum_{i=1}^{n} a_i * 10^{n-i}$, $\sum_{i=1}^{m} b_i * 10^{m-i}$. Если $n=m$, сравнение очевидно. Если $n<m$, то сравниваются $\sum_{i=1}^{n} a_i * 10^{n-i}$ и $\sum_{i=1}^{n} b_i * 10^{n-i}$, если и они равны – компаратор рекурсивно вызывается для $\sum_{i=n+1}^{m} a_n * 10^{m-i}$ и $\sum_{i=n+1}^{m} b_i * 10^{m-i}$, сравнивая оставшиеся части числа (запись для числа a означает, что последний разряд a_n повторяется $m-n+1$ раз, чтобы длины сравниваемых чисел совпадали).

Код решения:

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

using namespace std;

bool compare(const string &left, const string &right) {
  if (left.length() < right.length()) {
    if (left != right.substr(0, left.length())) {
      return left > right.substr(0, left.length());
    }
    return compare(right.substr(0, left.length()),
                   right.substr(left.length(), right.length()));
  } else if (left.length() > right.length()) {
    return !compare(right, left);
  }
  return left > right;
}
int main() {
  ifstream infile("number.in");
  string s;
  vector<string> v;
```

```
  while (infile >> s) {
    v.push_back(s);
  }
  sort(begin(v), end(v), compare);
  for (string s : v) {
    cout << s;
  }
  return 0;
}
```

Задача №G “Кошмар в замке”

Решение за O(n log n).
Задача заключается в максимизации веса строки s который вычисляется так: $A=\{a_i|a_i\in s\}, S_{max}=\sum_{i=1}^{|A|} c_i * max(\{d_{ij}\}_{i=1,\,j=1,a_i=a_j,i\neq j}^{|A|,|A|})$, где A - набор уникальных $a\in s$, c_i - вес a_i , d_{ij} - расстояние между a_i и a_j , $a_i=a_j$.
Для того, чтобы получить S_{max} необходимо, чтобы $a_i, a_j\in A: a_i=a_j$ с максимально возможным c_i находились на максимально возможном расстоянии друг от друга.
Рассмотрим так же $K=\{k_i|k_i=count(a_i), a_i\in S\}$, где k - количество a в S .
Докажем следующие утверждения:
1. $\forall a_i, k_i=1: c_i * d_{ij}=0$
Поскольку $k_i=1$, т.е. символ a встречается в s всего 1 раз, то $i=j$, следовательно, $d_{ij}=|i-j|=0$, отсюда понимаем, что при $k_i=1$, символ a не привносит ничего в сумму.
2. $\forall a_i: k_i>2, i_1, i_{k_i}, i_1 \ll i_{k_i}: \forall j: j=\{i_1+1 \dots i_{k_i}-1\}, i_1<j<i_{k_i}: max(\{d_{ij}\})_{i=1,\,j=1,a_i=a_j,i\neq j}^{|A|,|A|}=d_{i_1 i_{k_i}}$
Докажем при $k_i=3$:
Пусть $\exists i_1, i_2, i_3: i_1<i_2<i_3$.
Тогда $d_{12}=i_2-i_1, d_{23}=i_3-i_2, d_{13}=i_3-i_1$. Заметим, что поскольку $i_1<i_2$ и $i_2<i_3$ то по свойству транзитивности $i_1<i_3$, следовательно, $d_{12}<d_{13}$ и $d_{23}<d_{13}$.
Отсюда $max(\{d_{12}, d_{23}, d_{13}\})=d_{13}$ НУО верно для всех $k_i>2$ чтд.
Из (1) и (2) следует, что:
- символы с $k_i=1$ следует поставить в середину строки
- из символов с $k_i>2$ следует выделить 2 элемента для сортировки, остальные поставить в середину строки.

Тогда задача сводится к отделению элементов по (1) и (2), отправку остальных пар на сортировку по c_i , затем к поочередному добавлению элеметов в начало и конец строки.
Сортировка работает за $O(n \log n)$.

Код решения:

```
#include <algorithm>
#include <fstream>
#include <iostream>
#include <unordered_map>
    1. #include <utility>
#include <vector>

using namespace std;

int main() {
  ifstream infile("aurora.in");
  unordered_map<char, int> c;
  unordered_map<char, int> k;
  string s;
  infile >> s;
  for (char a : s)
    ++k[a];
  int w;
  for (char a = 'a'; a <= 'z'; ++a) {
    infile >> w;
    c[a] = w;
  }
  vector<pair<char, int>> vp;
  unordered_map<char, int> used;
  string res;
  for (char a : s) {
    if ((k[a] > 1) && (used[a] < 2)) {
      vp.push_back(make_pair(a, c[a]));
      ++used[a];
    } else {
```

```
      res += a;
    }
  }
  // stable sort
  sort(vp.begin(), vp.end(),
       [](const pair<char, int> a, const pair<char, int> b) {
         return a.second == b.second ? a.first < b.first : a.second < b.second;
       });
  for (int i = 0; i < vp.size(); i += 2) {
    res = vp[i].first + res;
    res += vp[i + 1].first;
  }
  cout << res;
  return 0;
}
```

Задача №H “Магазин”

Решение за O(n log n).

Что нам дает деление товаров по чекам? Для того, чтобы сделать результирующую сумму как можно меньше, необходимо удалить как можно больше товаров как можно большей суммы.
Отсортируем покупки по убыванию цены и получим массив $\{a_1, a_2, \dots a_n\}$ где $a_1 \geq a_2 \geq \dots \geq a_n$. Пусть $f(\{a_i \dots a_j\})$ - функция, принимающая чек от i до j товара и возвращающая товары, которые участвуют в акции. Пусть $k = \lfloor \frac{n}{2} \rfloor$ (для простоты пусть $n : 2$). Тогда:

- если не делить по чекам, то $f(\{a_1 \dots a_n\})$ = $\{a_{n-1}, a_n\}$. Так как эти 2 цены являются минимальными, это не самый подходящий вариант.
- если разделить на чек 1 и чек 2, то:
  - до тех пор, пока $n - i + 1 < k$, чек 2 $\{a_i \dots a_n\}$ в акции не участвует. В свою очередь чек 1 $\{a_1 \dots a_{i-1}\}$ такой, что $k \leq i - 1 < 2k$ позволит исключить 1 товар цены a_{i-1} , который даже может быть больше $a_{n-1} + a_n$. Заметим, что $\lim\limits_{i \to +(k+1)} f(\{a_1 \dots a_{i-1}\}) = a_k$

- как только $i-1=k$, видим, что и чек 1, и чек 2 участвуют в акции, тогда $\{f(\{a_1 \dots a_k\}), f(\{a_{k+1} \dots a_n\})\} = \{a_k, a_n\}$.
- дальнейший сдвиг $i-1<k$ не имеет смысла, так как тогда чек 1, имеющий самые дорогие товары, в акции участвовать не будет.

Из этих наблюдений заключаем, что a_k - самый дорогой товар, который можно исключить. Замечаем так же, что поскольку $\frac{n}{2}=k$, то набор исключенных товаров $\{a_k, a_{2k}\}$. Для более общего случая – если $\lfloor \frac{n}{m} \rfloor = k$, то необходимо исключить товары $\{a_k, a_{2k}, \dots a_{mk}\}$ для получения максимальной выгоды.
Вывод - n товаров необходимо отсортировать в порядке возрастания и разделить на $\lceil \frac{n}{k} \rceil$ чеков, где первые $\lfloor \frac{n}{k} \rfloor$ чеков будут иметь ровно k товаров.

Код решения:

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

using namespace std;

int main() {
  ios_base::sync_with_stdio(false);
  cin.tie(NULL);

  int n, k;
  cin >> n >> k;
  vector<int> a;
  int num, s = 0;
  for (int i = 0; i < n; ++i) {
    cin >> num;
    s += num;
    a.push_back(num);
  }
```

```
  if (k > n) {
    cout << s;
    return 0;
  }

  sort(a.begin(), a.end(), [](const int a, const int b) { return a > b; });
  int count = 0;
  for (int num : a) {
    ++count;
    if (count >= k) {
      s -= num;
      count = 0;
    }
  }
  cout << s;
  return 0;
}
```