Всероссийская олимпиада школьников по информатике
Региональный этап, I тур
Разбор задач

Андрианов И. А.
Стрекаловский О. А.

Вологодский государственный университет
Факультет прикладной математики,
компьютерных технологий и физики

Вологда
2016 г.

Классический алгоритм поиска второго максимума в массиве.

«Призы»

- Будем поддерживать две переменные: текущий максимум в массиве и второй по величине элемент.
- Тогда при добавлении в рассмотрение очередного приза обновим при необходимости первый и второй максимум.

«Призы»

```
1  int n = readInt();
2  int a = readInt();
3  int b = readInt();
4  int max = max(a, b);
5  int secondMax = min(a, b);
6  print(secondMax + "␣");
7  for (int i = 3; i <= n; i++) {
8      int num = readInt();
9      if (num > max) {
10         print(max + "␣");
11         secondMax = max;
12         max = num;
13     } else if (secondMax < num) {
14         print(num + "␣");
15         secondMax = num;
16     } else {
17         print(secondMax + "␣");
18     }
19 }
```

«Призы»

Решение на C++ с multiset:

```
1  int n;
2  std::cin >> n;
3  std::multiset<int> ma;
4  int x, y;
5  std::cin >> x >> y;
6  ma.insert(x);
7  ma.insert(y);
8  std::cout << *ma.begin() << "␣";
9  for (int i = 3; i <= n; i++) {
10     std::cin >> x;
11     ma.insert(x);
12     ma.erase(ma.begin());
13     std::cout << *ma.begin() << "␣";
14 }
```

«Призы»

Типичные ошибки

- Неэффективное решение.
  Много проходов по исходным данным в поисках второго максимума.
- Второй максимум — это не всегда предыдущий максимум, увиденный при проходе по массиву.
  Тест: 1 3 2.

Вопросы по задаче?

«Космическое поселение»

Основные идеи

- Перебор двух вариантов ориентации модулей.
- Двоичный поиск по величине толщины защиты жилого отсека при фиксированной ориентации модулей.
- Контроль переполнения 64-х разрядного типа данных.

«Космическое поселение»

- Пусть модуль ориентирован так, что сторона длиной a ориентирована вдоль стороны поля длиной w.
- Тогда после установки защиты толщиной d вдоль этой стороны можно установить $\lfloor w/(a+2d) \rfloor$ модулей.
- Аналогично, вдоль другой стороны можно установить $\lfloor h/(b+2d) \rfloor$ модулей.
- Всего можно установить $(\lfloor w/(a+2d) \rfloor) \times (\lfloor h/(b+2d) \rfloor)$ модулей.

«Космическое поселение»

Пример решения без переполнения 64–х разрядного типа

```
long long L = 0;
long long R = min( (w - a) / 2, (h - b) / 2) + 1;
while (L + 1 < R) {
    long long M = (L + R) / 2;
    long long ad = a + 2 * M;
    long long bd = b + 2 * M;
    long long ac = w / ad;
    long long bc = h / bd;
    if (ac * bc >= n) {
        L = M;
    } else {
        R = M;
    }
}
return L;
```

«А. С. Станкевич. Лекция 2: Сортировка и поиск».[1]

[1] http://www.youtube.com/watch?v=qkLLcdgJj_o

«Космическое поселение»

Типичные ошибки

- Неэффективное решение.
  Слишком большой перебор величины защиты.
- Переполнение 64-х разрядного типа данных при выборе слишком большой правой границы поиска или при промежуточных операциях.

«Космическое поселение»

Типичные ошибки

- Неэффективное решение.
  Слишком большой перебор величины защиты.
- Переполнение 64-х разрядного типа данных при выборе слишком большой правой границы поиска или при промежуточных операциях.

Решение на Java

В Java 8 появились методы:

- `Math.addExact(long x, long y)`
- `Math.multiplyExact(long x, long y)`

Кидают `ArithmeticException` в случае переполнения при выполнении операции.

Вопросы по задаче?

«Странные строки»

- Строка z является странной, если для нее выполняются следующие свойства:
  - z содержит не более двух различных символов;
  - все одинаковые символы в z идут подряд.

«Странные строки»

- Строка z является странной, если для нее выполняются следующие свойства:
  - z содержит не более двух различных символов;
  - все одинаковые символы в z идут подряд.
- Таким образом, для определения странности строки требуется найти количество ее подстрок, которые имеют такую структуру.
  Сделаем это в два этапа.

Если в строке длиной N встречается только один символ, то её странность равна N.

- Пусть в строке встречаются подстроки из одного и того же символа.

- Пусть в строке встречаются подстроки из одного и того же символа.
- Пусть таких подстрок K и их размеры $L_1, L_2, \ldots, L_K$.

«Странные строки»
Этап 1. Подстроки из повторяющихся символов

- Пусть в строке встречаются подстроки из одного и того же символа.
- Пусть таких подстрок K и их размеры $L_1, L_2, \ldots, L_K$.
- Тогда каждый такой символ добавляет к странности исходной строки $\max L_i$, где $1 \le i \le K$.
  Пример: $\ldots \underbrace{aaaa}_{4} \ldots \underbrace{aa}_{2} \ldots$
  - Подстроки «$aaaa$» и «aa» являются странными.
  - Из них можно выделить подстроки, которые являются подстроками исходной строки: «a», «aa», «aaa», «$aaaa$».
  - В итоге символ «a» создаёт 4 подстроки и добавляет 4 к общей странности строки.

- Будем решать задачу независимо для каждой упорядоченной пары (a, b) символов.

«Странные строки»

Этап 2. Подстроки из двух разных символов

- Будем решать задачу независимо для каждой упорядоченной пары (a, b) символов.
- Пусть в строке есть подстроки вида $X \underbrace{aa\ldots a}_{k_i}\underbrace{bb\ldots b}_{l_i} Y$, где множество X — любой символ $\neq a$ (или начало строки), а Y — любой символ $\neq b$ (или конец строки).

«Странные строки»

Этап 2. Подстроки из двух разных символов

- Будем решать задачу независимо для каждой упорядоченной пары (a, b) символов.
- Пусть в строке есть подстроки вида $X \underbrace{aa\ldots a}_{k_i}\underbrace{bb\ldots b}_{l_i} Y$, где множество X — любой символ $\neq a$ (или начало строки), а Y — любой символ $\neq b$ (или конец строки).
- Поместим все пары (k_i, l_i) в массив.

«Странные строки»

Этап 2. Подстроки из двух разных символов

- Будем решать задачу независимо для каждой упорядоченной пары (a, b) символов.
- Пусть в строке есть подстроки вида $X \underbrace{aa\ldots ab}_{k_i}\underbrace{b\ldots b}_{l_i}Y$, где множество X — любой символ $\neq a$ (или начало строки), а Y — любой символ $\neq b$ (или конец строки).
- Поместим все пары (k_i, l_i) в массив.
- Теперь нам надо найти число пар (k, l), таких, что найдется (k_i, l_i), такая что $1 \le k \le k_i$ и $1 \le l \le l_i$.

- Для поиска числа таких пар отсортируем все пары по k_i, а затем по l_i.

- Для поиска числа таких пар отсортируем все пары по k_i, а затем по l_i.
- Для каждой пары (k_i, l_i) заменим l_i на максимум l_j, где $k_j \geq k_i$.

«Странные строки»

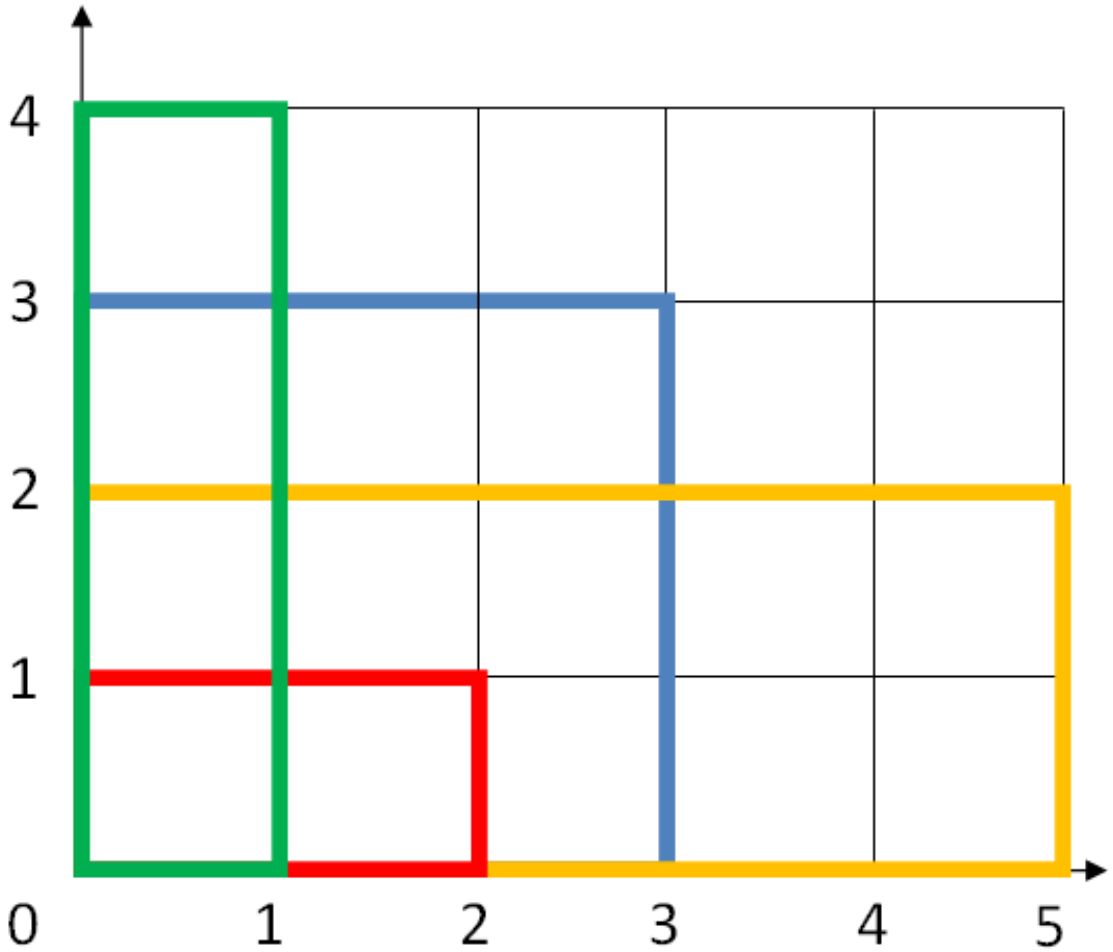
Этап 2. Подстроки из двух разных символов

- Для поиска числа таких пар отсортируем все пары по k_i, а затем по l_i.
- Для каждой пары (k_i, l_i) заменим l_i на максимум l_j, где $k_j \geq k_i$.
- Теперь оставим по одной паре с каждым k_i и просуммируем величины $(k_i - k_{i-1}) \times l_i$ для всех i.

- Для поиска числа таких пар отсортируем все пары по k_i, а затем по l_i.
- Для каждой пары (k_i, l_i) заменим l_i на максимум l_j, где $k_j \geq k_i$.
- Теперь оставим по одной паре с каждым k_i и просуммируем величины $(k_i - k_{i-1}) \times l_i$ для всех i.
- Получившееся значение добавим к ответу на задачу.

«Странные строки»

Это алгоритм поиска площади объединения прямоугольников с осями, параллельными осям координат, общим углом в $(0, 0)$ и положительными координатами противоположного угла.



«Странные строки»

Типичные ошибки

- Неэффективное решение.
  Проверка подстрок на странность «в лоб».
- Неэффективное удаление дубликатов с помощью вставки строк в структуру данных «множество».

Вопросы по задаче?

«Поездка на каникулах»

Жадный алгоритм, дающий наименьшее число пересадок:

- Покупаем билет до станции назначения, если это возможно, иначе покупаем билет до самой дальней возможной станции. Для этой станции действуем аналогично, и т.д.
- Пояснение: незачем заранее пересаживаться на другое место, пока свободно прежнее

«Поездка на каникулах»

1. Пусть `go1[i]` — максимальный номер станции, до которой можно доехать от станции i, используя ровно 1 билет.
2. Допустим, массив `go1` каким–то образом построен, и нужно проехать от f до s.
3. Тогда билеты надо покупать на станциях f, `go1[f]`, `go1[go1[f]]` и т.д.
4. Обработка одного варианта поездки при этом выполняется за $O(N)$ — то есть при условии эффективного построения массива `go1` решаем первые две подзадачи.

- Наивное построение — квадратичная или более высокая вычислительная сложность.
  Позволяет решить лишь подзадачу 1.
- Более эффективное построение:
  Будем говорить, что место a свободно на протяжении отрезка $[c, d]$, если никакой из проданных билетов не занимает это место между станциями c и d.

Построим все максимальные свободные отрезки для каждого места:

1. Вначале для каждого места имеется один свободный отрезок — от 1 до n.
2. Отсортируем все билеты с данным местом по станции отправления.
3. Будем рассматривать билеты в этом порядке (удобно представлять, что их в таком порядке покупали).
4. Каждый следующий билет вырезает кусок из последнего отрезка, расщепляя его на два.

Пример

Пусть $N = 10$, рассматриваем место №1.

1. Вначале место №1 свободно на всём пути – один отрезок $[1, 10]$.
2. Куплен билет от 2 до 5 — стало 2 отрезка: $[1, 2]$, $[5, 10]$.
3. Куплен билет от 8 до 9 — стало 3 отрезка: $[1, 2]$, $[5, 8]$, $[9, 10]$.
4. ...

«Поездка на каникулах»
Пример реализации шага №1

```
1  std::sort(tickets.begin(), tickets.end());
2  std::vector<std::vector<Range> > r(nPlaces);
3  for (int i = 0; i < nPlaces; i++) {
4      r[i].push_back(Range(0, n - 1));
5  }
6  for (Ticket &t : tickets) {
7      r[t.place].back().to = t.from;
8      if (t.to != n - 1) {
9          r[t.place].push_back(Range(t.to, n - 1));
10     }
11 }
```

Объединим множества свободных отрезков всех мест и отсортируем их по начальной станции.

Пример

Пусть у нас есть два места:

- место №1 — отрезки $[1, 4]$, $[6, 8]$;
- место №2 — отрезки $[1, 7]$, $[8, 9]$;

Результат объединения и сортировки массива отрезков: $[1, 7]$, $[1, 4]$, $[6, 8]$, $[8, 9]$.

«Поездка на каникулах»
Пример реализации шага №2

```
1 std::vector<Range> ranges;
2 for (int i = 0; i < nPlaces; i++) {
3     std::copy(
4             r[i].begin(),
5             r[i].end(),
6             back_inserter(ranges)
7     );
8 }
9 std::sort(ranges.begin(), ranges.end());
```

Построим все максимальные свободные отрезки для каждого места:

1. Проходим по всем станциям и одновременно по полученному массиву отрезков, при этом храним и обновляем самый правый конец отрезка, который у нас встречался.
2. На каждом шаге записываем это значение в массив go1 (но только если оно больше текущей станции).

«Поездка на каникулах»
Построение массива go1. Шаг №3

Пример

Полученный ранее массив отрезков: $[1, 7]$, $[1, 4]$, $[6, 8]$, $[8, 9]$.

- Станция №1. В массиве проходим по двум первым элементам, на третьем остановились. Самый правый конец = 7, то есть go1[1] = 7.
- Станция №2. В массиве остаёмся на том же месте. Самый правый конец всё еще 7, то есть go1[2]=7.
- По аналогии go1[3] = go1[4] = go1[5] = 7
- Станция №6. В массиве проходим по следующим двум элементам, самый правый конец = 9. Отсюда go1[6]=9.
- По аналогии go1[7] =9, go1[8]=9.
- Поскольку самый правый конец = 9 — не больше номера станции, то go1[9] = 0. Аналогично, go1[10]=0.

«Поездка на каникулах»
Пример реализации шага №3

```
1  std::vector<int> go1(n);
2  int rightmost = 0,  range = 0;
3  for (int i = 0; i < n; i++) {
4      while (range < (int)ranges.size()
5                && ranges[range].from == i) {
6          rightmost = std::max(
7                  rightmost, ranges[range].to);
8          range++;
9      }
10     if (rightmost > i) {
11         go1[i] = rightmost;
12     }
13 }
```

Обратите внимание на сходство с алгоритмом вычисления z-функции для строк

Будем поддерживать множество «открытых» свободных отрезков, которые начинаются не позже рассматриваемой станции, в структуре данных, которая позволяет искать максимум по ключу (например, двоичная куча, дерево отрезков или `std::set`).
В качестве ключа используем конец отрезка.

1. Рассматривая очередную станцию i, добавим все свободные отрезки, которые в ней начинаются, в множество открытых отрезков.
2. Теперь `go1[i]` равно максимальному концу открытого отрезка, либо i, если множество открытых отрезков пусто, или все они заканчиваются до i.

Нужно на каждый запрос отвечать быстрее, чем за $O(n)$.
Варианты:

- Sqrt–декомпозиция — $O(\sqrt{(n)})$;
- Метод двоичных подъёмов — $O(\log(n))$;

1. Для каждой станции i делаем $\sqrt{(n)}$ прыжков по массиву `go1` (то есть $i \to$ `go1(i)` $\to$ `go1(go1(i))` $\to \ldots$) и запоминаем, где мы оказались.
2. Пусть `skip[i]` — станция, в которую мы попадём, если сделать $\sqrt{(n)}$ прыжков по массиву `go1`.

Теперь мы умеем делать не только малые прыжки длиной 1, но и большие прыжки длиной $\sqrt{(n)}$.
Ответ на очередной запрос ищется так:

1. Вначале совершаем большие прыжки, пока это возможно и пока не выйдем за конечную станцию.
2. Возвращаемся к предыдущему шагу и совершаем малые прыжки, пока не придём в конечную станцию или станцию после неё.

$\sqrt{(200000)} \approx 450$
Максимальное количество действий порядка
$2 \times 200000 \times 450 = 180$ миллионов

«Поездка на каникулах»

Пример заполнения массива skip и поиска ответа на запрос

```
1  int SKIP = sqrt(n);
2  std::vector<int> skip(n);
3  for (int i = 0; i + SKIP < n; i++) {
4      int j = i, k;
5      for (k = 0; k < SKIP && go1[j] > 0; k++)
6          j = go1[j];
7      if (k == SKIP)
8          skip[i] = j;
9  }
10 // поиск ответа
11 while (go1[from] < to) {
12     if (skip[from] > 0 && skip[from] < to) {
13         answer += SKIP;
14         from = skip[from];
15     } else {
16         answer++;
17         from = go1[from];
18     }
19 }
```

Пусть `go[i][j]` равна максимальному номеру станции, до которого можно доехать от станции j, используя не более 2^i билетов.

Алгоритм построения матрицы go основан на следующей идее:

1. Пусть можно от j доехать максимум до u, используя не более 2^{i-1} билетов.
2. Пусть от u можно доехать максимум до v, используя не более 2^{i-1} билетов.
3. Тогда от j можно доехать до v, используя не более 2^i билетов.

«Поездка на каникулах»
Пример заполнения массива go

```
1 for (int i = 1; i <= sz; i++) {
2     for (int j = 0; j < n; j++) {
3         int u = go[i - 1][j];
4         go[i][j] = go[i - 1][u];
5     }
6 }
```

В качестве sz следует выбрать минимальную величину, такую что $2^{sz} \geq n$.

Запрос на поездку от fr до to

```
1  if (go[sz][fr] < to) {
2      cout << -1 << endl;
3  } else {
4      int steps = 0;
5      int curK = sz;
6      while (go1[fr] < to) {
7          while (go[curK][fr] >= to) {
8              curK--;
9          }
10         steps += 1 << curK;
11         fr = go[curK][fr];
12     }
13     cout << steps + 1 << endl;
14 }
```

Вопросы по задаче?

Спасибо за внимание!