III Областная олимпиада школьников по информатике
2018-2019 учебный год
9-10 классы
Отборочный тур

Разбор задач

Задача 1. Собачий приют (10 баллов)

Количество собак в самом заполненном вольере можно получить как округление в большую сторону числа *N* / *K*. Используя допустимый набор операций, это можно сделать так:

$$(N + K - 1) / K$$

(помним, что, согласно условию задачи, '/' означает деление нацело).

Теперь построим вторую формулу. Число вольеров с максимальным числом собак равно *N* % *K*. Тогда число вольеров, в которых сидит на одну собаку меньше, равно *K* – *N* % *K*. Однако, данная формула даст неверный ответ, когда *N* делится на *K* – должно быть 0, а формула выдаст *K*. Чтобы это исправить, достаточно добавить остаток от деления на *K*:

$$(K - N \% K) \% K$$

Можно рассуждать немного по-другому. Нам нужно, чтобы выражение *N* % *K* давало *K* вместо нуля, если *N* делится на *K*. Для этого достаточно *N* % *K* заменить на *(N - 1) % K + 1*, тогда окончательно получаем:

$$K - ((N - 1) \% K + 1)$$

Поскольку в задаче возможны различные верные ответы, то проверку решений нельзя выполнять простым сравнением файлов – проверка осуществляется подстановкой заданного ряда тестовых значений *N* и *K* в формулу участника. При этом возможно, что даже неправильные формулы на некоторых тестовых данных выдадут верный результат, и решение получит частичные баллы.

Задача 2. Очередь (10 баллов)

Заменим числа 50 на открывающие скобки, а 100 – на закрывающие. Теперь задачу можно переформулировать так: определить количество различных «почти правильных» скобочных последовательностей из *N* открывающих и *M* закрывающих скобок. «Почти правильной» скобочной последовательностью будем называть

последовательность, которую можно достроить до правильной путём дописывания закрывающих скобок в конец.

Пусть $f[i][j]$ – количество «почти правильных» скобочных последовательностей с i открывающими и j закрывающими скобками. Вначале выделим пару частных случаев:

1). $f[i][j] = 1$ при $j = 0$.

2). $f[i][j] = 0$ при $i < j$.

Перейдём теперь к общему случаю ($0 < j \leq i$). Попробуем получить рекуррентное соотношение для $f[i][j]$, то есть выразим $f[i][j]$ через f с меньшими индексами. Разобьём все последовательности из i открывающих и j закрывающих скобок на два множества: в первом множестве все последовательности заканчиваются открывающей скобкой, во втором – закрывающей. Размер первого множества – это $f[i - 1][j]$, размер второго – это $f[i][j - 1]$. Тогда:

$$f[i][j] = f[i - 1][j] + f[i][j - 1]$$

Пользуясь этой формулой, можно последовательно сосчитать значения f, двигаясь от меньших индексов к большим. Чтобы не считать вручную, можно, например, воспользоваться программой для работы с электронными таблицами. На рисунке показана часть такой таблицы в Microsoft Excel. Первый столбец заполнен единицами, остальные ячейки вычисляются как сумма ячеек на 1 выше и на 1 левее. Цветом выделены ответы на первые три вопроса задачи.

	0	1	2	3	4	5	6	7
0	1							
1	1	1						
2	1	2	2					
3	1	3	5	5				
4	1	4	9	14	14			
5	1	5	14	28	42	42		
6	1	6	20	48	90	132	132	
7	1	7	27	75	165	297	429	429
8	1	8	35	110	275	572	1001	1430
9	1	9	44	154	429	1001	2002	3432
10	1	10	54	208	637	1638	3640	7072
11	1	11	65	273	910	2548	6188	13260
12	1	12	77	350	1260	3808	9996	23256
13	1	13	90	440	1700	5508	15504	38760
14	1	14	104	544	2244	7752	23256	62016
15	1	15	119	663	2907	10659	33915	95931

Ещё один способ решения – написать программу, выполняющие описанные вычисления. Пример такой программы на C++:

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

std::vector<std::vector<long long> > a(31, std::vector<long
long>(21));

void solve(int n, int m) {
  a[0][0] = 1;
  for (int i = 1; i <= n; i++) {
    for (int j = 0; j <= std::min(m, i); j++) {
      if (i == 0 && j == 0)
        continue;
      if (i > j) {// (
        a[i][j] += a[i - 1][j];
        assert(a[i][j] >= 0);
      }
      if (j > 0) { // )
        a[i][j] += a[i][j - 1];
        assert(a[i][j] >= 0);
      }

    }
  }
}

int main() {
  solve(30, 20);
  std::cout << a[4][2] << std::endl;
  std::cout << a[5][3] << std::endl;
  std::cout << a[6][6] << std::endl;
  std::cout << a[20][10] << std::endl;
  std::cout << a[30][20] << std::endl;
}
```

Ответы на первые четыре вопроса задачи можно было получить ещё одним способом – написать программу, генерирующую все возможные допустимые последовательности и подсчитывающую их количество. Пример такого решения есть в архиве жюри (файл queue_bf.cpp).

Задача 3. Шарики (10 баллов)

Всё, что требуется сделать в этой задаче – выполнить аккуратную программную реализацию алгоритма добавления шариков, описанного в условии. Вероятно, задача не вызовет сложностей у большинства участников.

Остановимся подробнее на одном моменте. Поскольку шарики нужно добавлять и удалять с обоих концов, нам необходима структура данных, позволяющая это делать эффективно. В некоторых языках программирования подобная структура имеется в стандартной библиотеке (например, std::deque в C++). Однако, несложно такую структуру реализовать и самостоятельно на базе массива: создаём

массив длины $2 \cdot N$ и первый шарик кладём в середину массива (в позицию N) – тогда нам хватит места в массиве, чтобы добавлять оставшиеся N-1 шарик хоть влево, хоть вправо. Пример такой реализации на языке C++:

```
#include <stdio.h>
#include <vector>

int main() {
  int n;
  scanf("%d\n", &n);
  std::vector<int> v(2 * n);
  int left = -1, right = -1;
  for (int i = 0; i < n; i++) {
    char cmd;
    int x;
    scanf("%c %d\n", &cmd, &x);
    if (left < 0) {
      left = right = n;
      v[left] = x;
    }
    else if (cmd == '2') {
      if (right - left >= 1 && v[right] == x && v[right - 1] == x)
        right -= 2;
       else
        v[++right] = x;
    }
    else if (cmd == '1') {
      if (right - left >= 1 && v[left] == x && v[left + 1] == x)
        left += 2;
      else
        v[--left] = x;
    }
  }
  printf("%d\n", right - left + 1);
}
```

Задача 4. Подстрока (10 баллов)

Вначале опишем простой способ решения подзадач 1 и 2. Найдём для каждой позиции i входной строки минимальную длину подстроки, которая начинается в позиции i и содержит все гласные буквы. Для этого во вложенном цикле будем идти вправо от позиции i и подсчитывать, сколько раз встретилась каждая буква (проходим очередную букву – увеличиваем соответствующий счётчик). Как только каждая гласная буква встретилась хотя бы одному разу, очередная длина найдена. Среди всех таких длин нужно выбрать минимальную. Количество строк минимальной длины можно подсчитывать попутно по ходу алгоритма.

Сложность данного решения составляет $O(N^2)$.

Пример реализации на языке C++:

```
#include <iostream>
#include <string>

inline bool is_vowel(char c) {
 return c=='a' || c=='e' || c=='i' || c=='o' || c=='u';
}

int main() {
  std::string s;
  std::cin >> s;
  int n = s.length();
  s = "$" + s;
  int ans = 1000000000, count = 0;
  for (int i = 1; i <= n; i++) {
    if (!is_vowel(s[i])) continue;
    bool c[128] = {};
    for (int j = i; j <= n; j++) {
      if (is_vowel(s[j])) {
        c[(int)s[j]] = true;
        if (c['a'] && c['e'] && c['i'] && c['o'] && c['u']) {
          int len = j - i + 1;
          if (len < ans) {
            ans = len; count = 1;
          } else if (len == ans) {
            count++;
          }
          break;
        }
      }
    }
  }
  std::cout << ans << " " << count << std::endl;
}
```

Полное решение данной задачи основано на использовании техники двух указателей (слово «указатель» в данном контексте означает индекс элемента в массиве).

Создадим два указателя – left и right. Вначале они будут ссылаться на позицию перед первым элементом строки.

В цикле будем двигать указатель right вправо, увеличивая счётчик для каждой встреченной буквы. Как только счётчики всех гласных букв станут больше нуля, останавливаемся. Мы нашли строку от *left* до *right*, содержащую все гласные буквы.

Теперь попробуем укоротить эту строку с левого края, двигая вправо указатель *left* и уменьшая счётчик каждой встреченной буквы. Как только счётчик какой-то гласной буквы обнулился, останавливаемся.

Найденная в итоге строка *s*[*left*..*right*] содержит все гласные буквы, и её нельзя укоротить ни с левого края, ни с правого.

Сравниваем её длину с текущим минимум и при необходимости его обновляем.

Далее процесс продолжается аналогично – снова двигаем указатель *right*, затем *left*, и т.д. Количество подстрок минимальной длины можно подсчитывать попутно по ходу работы.

Сложность данного решения составляет $O(N)$.

Пример реализации на C++:

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

inline bool match(const std::vector<int> &c) {
  return c['a'] && c ['e'] && c['i'] && c['o'] && c['u'];
}

int main() {
  std::ios_base::sync_with_stdio(0);
  std::string s;
  std::cin >> s;
  int n = s.length(), ans = n + 1, count = 0;
  std::vector<int> c(128);
  int left = -1, right = -1;
  for (;;) {
    while (right + 1 < n) {
      c[s[++right]]++;
      if (match(c)) break;
    }
    if (!match(c)) break;
    for(;;) {
      c[s[++left]]--;
      if (!match(c)) {
        int len = right - left + 1;
        if (len < ans) {
          ans = len;
          count = 1;
        } else if (len == ans) {
          count++;
        }
        break;
      }
    }
  }
  if (ans > n)
    std::cout << -1 << std::endl;
  else
    std::cout << ans << " " << count << std::endl;
}
```

Ещё один способ решения данной задачи основан на методе бинарного поиска. Вначале выполним следующий преподсчёт: для каждой гласной буквы и позиции i определим, сколько раз эта буква встречалась в позициях $\leq i$. Сохраним эти значения в матрице p.

Например, для строки “redautomobile” p[‘o’][12] = 2, так как буква ‘o’ дважды встречалась слева от позиции 12. Такой преподсчёт несложно выполнить за линейное время.

Теперь для любой подстроки и любой гласной буквы мы можем определить, сколько раз данная буква встретилась в данной подстроке, сделав всего одну операцию вычитания: количество букв a в подстроке s[$left$..$right$] равняется p[a][$right$] – p[a][$left$ – 1].

Теперь с помощью бинарного поиска подберём ответ. Взяв очередную длину, мы можем за один проход по строке проверить, существует ли подстрока такой длины, содержащая все гласные буквы.

Сложность такого решения составляет $O(N \cdot \log(N))$, что также позволяет набрать полный балл за эту задачу. Пример похожего решения имеется в архиве жюри (файл substring_n2.cpp).