Задача C (7-8), A (9-11). Лифты

Автор задачи: Тимур Хисматуллин
Подготовка тестов: Марк Рябов, Николай Макеенков

Первый вариант решения — решение перебором. Можно было пройти циклом от 1 до N и проверить, достает ли Вася до кнопки этого этажа, и после проверить выгоднее ли ему ехать именно через этот этаж. И в конце вывести тот этаж, через который ехать меньше. Данное решение работает за $O(N)$, поэтому оно не набирало полный балл, так как не укладывалось в отведенное ограничение в 1 секунду.

Второе решение, которое набирало полный балл состояло из рассмотрения трех случаев:

- Вася достает до кнопки нужного этажа;
- Васе выгднее ехать с ближайшего снизу этажа, до которого он дотягивается;
- Васе выгодно ехать с ближайшего сверху этажа, до которого он дотягивается. В этом случае нужно проверить, существует ли этот этаж.

Сначала найдем, в каком столбце находится нужный Васе этаж. Если в первом столбце находятся этажи с 1 по T-ый, во втором с $(T+1)$ по $2T$-ый, и так далее, то несложно вывести формулу для номера столбца: $S = \left\lfloor \frac{K-1}{T} \right\rfloor$ (столбцы в таком случае нумеруются с 0).

После этого несложно найти номер кнопки Васиного этажа в столбце, она равна $Z = K - S \cdot T$. Тогда

1. если $Z \leqslant L$, то Вася может нажать кнопку своего этажа и сразу приехать куда нужно;
2. если $Z > L$ и $Z - L$ (расстояние, которое Вася должен пройти вверх) меньше, чем $T - Z + 1$ (расстояние, на которое Вася должен спуститься вниз), то Вася должен подниматься вверх;
3. если $Z > L$ и $Z - L \geqslant T - Z + 1$, то Вася должен спускаться вниз.

Приведем отрывок решения на языке Python:

```
S = (K - 1) // T
Z = K - S * T
if Z <= L:
        print(K)
elif Z - L < T - Z + 1:
        print(S * T + L)
else:
        print((S + 1) * T + 1)
```

Задача D (7-8), B(9-11). Торговля

Автор задачи: Иван Соломатин
Подготовка тестов: Иван Соломатин, Андрей Поповкин

Для решения данной задачи существует два способа:

1. Можно написать программу, которая будет перебирать значения p и q в пределах $[2, 10]$ и выводить такие p и q, что они удовлетворяют условию задачи.
   Более конкретно: фиксируем p и q. Переведём заданные числа в 10-чную систему счисления:

$$\begin{aligned} n_p &\to n_{10}, \\ m_q &\to m_{10}, \\ (n+m)_p &\to a_{10}, \\ (n+m)_q &\to b_{10}. \end{aligned}$$

   При этом нужно не забыть проверить, что введённые числа действительно могут быть числами в системах счисления с основаниями p и q. То есть, что все цифры числа меньше основания его предполагаемой системы счисления.
   Тогда p и q удовлетворяют условиям задачи, если:

$$n_{10} + m_{10} = a_{10} = b_{10}$$

   Фрагмент решения на C++ приведен ниже:

```
for (int p = 2; p <= 10; p++)
  for (int q = 2; q <= 10; q++) {
    int a = to10(n, p) + to10(m, q);
    int b = to10(nmp, p);
    int c = to10(nmq, q);
    if ((a > 0) && (b > 0) && (c > 0) && (a == b) && (b == c))
      cout << p << ' ' << q << endl;
  }
```

2. Для решения данной задачи не обязательно писать программу. Способ 2 подразумевает решение без использования языков программирования.
   По условию задачи, задано число n в системе счисления p и число m в системе счисления q, а так же их сумма в системах счисления p и q. Учитывая, что все числа в тестах двузначные, введём обозначения:

$$n_p = \overline{ab}_p; \;\; m_q = \overline{cd}_q; \;\; (n+m)_p = \overline{ef}_p; \;\; (n+m)_q = \overline{gh}_q,$$

   где запись $n_p = \overline{ab}_p$ обозначает, что число n в p-ичной системе счисления, получается приписыванием цифры b справа от цифры a.
   Переведём все числа в одну систему счисления, например, в 10-чную:

$$n_{10} = ap + b; \;\; m_{10} = cq + d; \;\; (n+m)_{10} = ep + f = gq + h.$$

   Отсюда получаем:

$$ap + b + cq + d = ep + f = gq + h.$$

   Разбиваем на два уравнения:

$$\begin{cases} ap + b + cq + d &= ep + f \\ ep + f &= gq + h \end{cases}.$$

   Так как в условии сказано, что решение есть и оно единственно, остаётся только разрешить эту систему относительно p и q.

Задача C (9-11). Пароль

Автор задачи: Николай Першаков
Подготовка тестов: Николай Першаков, Николай Макеенков

В задаче была дан целочисленный массив S длины n и было нужно найти наибольший его подмассив, не содержащий двух одинаковых подмассивов. Решение, которое просто перебирало все подмассивы S и для каждого проверяло, есть ли у него два равных подмассива, гарантированно набирало 12 баллов.

Для получения большего числа баллов нужно было заметить, что если у какого-то массива A есть два равных подмассива, то в A есть и два равных элемента (например, первые числа равных подмассивов). Также очевидно, что если есть два равных элемента, то и равные подмассивы найдутся. Таким образом задача переформулируется так: необходимо найти наибольший подмассив массива, не содержащий двух одинаковых элементов.

Решим задачу для зафиксированного левого конца подмассива. Заведем массив cnt, в котором $cnt[i]$ равно количеству вхождений числа i в текущий подмассив (cnt называется частотным словарем). Изначально все элементы этого словаря будут равны 0. Теперь будем последовательно удлинять текущий подмассив на один элемент. Если к нему добавился элемент x, то нужно увеличить $cnt[x]$ на один. Если же до этого прибавления оно было ненулевым, то мы получили массив, в котором есть две одинаковые буквы — то есть последнее удлинение было лишним и его надо отменить. Более длинные подмассивы с тем же левым концом рассматривать не имеет смысла, так как они тоже будут содержать те же два равных числа. Таким образом для фиксированной левой границы мы научились решать задачу за $O(n)$. Добавив перебор этой границы, получим решение всей задачи за $O(n^2)$, которое гарантированно набирало 36 баллов.

Заметим теперь, что если для левой границы l самый длинный подходящий нам подмассив заканчивается на позиции r, то при сдвиге левой границы на один элемент вправо, правая граница никак не может сдвинуться влево (в подмассиве с l по r не было одинаковых чисел, следовательно в подмассиве с $l+1$ по r их тем более не будет). Это позволяет нам при переходе от l к $l+1$ начинать поиск подходящей правой границы не с самого начала, а с предыдущего r (только важно не забыть уменьшить число вхождений $S[l]$ в частотном словаре). В таком решении и левая граница l, и правая r суммарно сдвинутся вправо не более n раз каждая. Каждый из таких сдвигов обрабатывается за $O(1)$. Таким образом получили решение, работающее за $O(n)$ и набирающее 100 баллов.

Приведем отрывок решения на языке Python:

```
res_l = res_r = r = 0
cnt = [0] * MAXN
cnt[a[r]] += 1
for l in range(0, n):
    if r < l:
        cnt[a[l]] += 1
        r = l
    while r < n - 1:
        r += 1
        cnt[a[r]] += 1
        if cnt[a[r]] > 1:
            cnt[a[r]] -= 1
            r -= 1
            break
    if r - l > res_r - res_l:
        res_r, res_l = r, l
    cnt[a[l]] -= 1
```

Задача D (9-11). Виртуоз гитары

Автор задачи: Иван Корябкин
Подготовка тестов: Дмитрий Кузьмичёв, Наиль Мингалиев

Воспользуемся методом динамического программирования. Будем хранить в трехмерном массиве $dp[i][a][b]$ значение true (истина), если мы можем сыграть первые i нот, при этом в конце исполнения первая струна будет настроена на ноту a, а вторая — на ноту b, и false (ложь), если не можем.

Тогда мы можем заметить, что из истинности $dp[i][a][b]$ вытекает, что мы также можем сделать переход в состояния $dp[i+1][a][b]$, $dp[i+1][a-1][b]$, $dp[i+1][a+1][b]$, $dp[i+1][a][b-1]$, $dp[i+1][a][b+1]$, донастраивая струны на одну ноту выше или ниже (не следует забывать, что мы за раз можем изменить только одну струну). Однако важно не забывать, что после перенастройки струн Петя должен быть в состоянии исполнить $(i+1)$-ю ноту, то есть она должна быть равна либо a, либо b.

Чтобы восстановить ответ нужно хранить дополнительно для каждого состояния вспомогательную информацию, из какого состояния произошел переход в него.

Приведем отрывок решения данной задачи на C++:

```
for (int i = 0; i < maxCh; i++)
  for (int j = 0; j < maxCh; j++)
    dp[0][i][j] = true;

for (int i = 1; i <= n; i++)
  for (int j = 0; j < maxCh; j++) {
    int x = a[i];
    int y = j;
    for (int t = 0; t <= 1; t++) {
      for (int dx = -1; dx <= 1; dx++)
        for (int dy = -1; dy <= 1; dy++)
          if (dx == 0 || dy == 0) {
            int nx = x + dx;
            int ny = y + dy;
            if (nx >= 0 && nx < maxCh && ny >= 0 && ny < maxCh
                && dp[i - 1][nx][ny] == 1) {
              dp[i][x][y] = 1;
              par[i][x][y] = pcc(nx, ny);
            }
          }
      swap(x, y);
    }
  }
```