Разборы задач муниципального этапа ВсОШ по информатике

профиль: программирование

2025

Оглавление

7-8

Билет в кино

Задача решается простым перебором вариантов.

134116

398299

274607

63255560

526716221499

Шахматы

Указание к решению, в решении важно учитывать все пересечения, для этого нужно понять, что на диагоналях сохраняется четность суммы строки и столбца (инвариант), поэтому, например, диагонали для ферзей в точке (1, 1) и (10, 1) не пересекаются.

Решение:

16

50

62

66

880

Урожай

Алгоритм решения. По числу надо составить систему, учитывая, что исходное число должно быть минимальным; решить, перебирая все варианты, начиная, с первой цифры.

Ответы на задачу:

12233

80000

25000

10580

39689

7-8, 9-11

Яблони

Решение

Все возможные яблони можно посадить на 4 участка со сторонами n // 2 - a + 1, n // 2 - b + 1, откуда следует решение задачи.

Листинг, Python

```
n = int(input())
a = int(input())
b = int(input())

if a > n // 2 or b > n // 2:
    print(0)
else:
    result = max(0, 4 * (n // 2 - a + 1) * (n // 2 - b + 1))
```

```
print(result)
```

Маша украшает дачу

Нужно проверить, что в массиве есть два либо ноль элементов, четность которых не совпадает с индексами, на которых они стоят. Если таких элементов два, то проверяем, что четности их разные, и меняем местами. Если таких элементов ноль, то меняем элементы, стоящие на 1 и 3 позициях. Частным случаем является случай при n = 2, для него невозможно произвести обмен в этом случае.

Решение , C++

```
#include <iostream>
#include <vector>
using namespace std;
using int_ = long long;
int main()
{
    int n;
    cin >> n;
    vector<int_> a(n);
    for (int i = 0; i < n; i++)
        cin >> a[i];
    vector<vector<int_>> cnt(2);
    for (int i = 0; i < n; i++)
        if (a[i] % 2 == i % 2)
            cnt[i % 2].push_back(i);
    if (cnt[0].size() == cnt[1].size() && cnt[0].size() <= 1)
    {
        if (cnt[0].size() == 0)
        {
            if (n < 3)
                cout << "-1 -1" << endl;
            else
                cout << "1 3" << endl;
```

```
        }
        else
        {
            cout << cnt[0][0] + 1 << ' ' << cnt[1][0] + 1 << endl;
        }
    }
    else
        cout << "-1 -1" << endl;
    return 0;
}
```

Решение (Python)

```
def main():
    n = int(input())
    # Вместо массивов используем счетчики и переменные для хранения индексов
    cnt0 = 0
    cnt1 = 0
    index0 = -1
    index1 = -1
    # Читаем и обрабатываем числа последовательно
    for i in range(n):
        a = int(input())
        # Проверяем условие a[i] % 2 == i % 2
        if a % 2 == i % 2:
            if i % 2 == 0:
                cnt0 += 1
                index0 = i
            else:
                cnt1 += 1
                index1 = i
    # Проверяем условия как в оригинальном коде
    if cnt0 == cnt1 and cnt0 <= 1:
        if cnt0 == 0:
```

```
            if n < 3:
                print("-1 -1")
            else:
                print("1 3")
        else:
            print(f"{index0 + 1} {index1 + 1}")
    else:
        print("-1 -1")
if __name__ == "__main__":
    main()
```

9-11

Саженцы

Решение

Решение работает за время O(n). Основная идея – использовать сортировку подсчетом, для этого:

1. **Инициализируем массивы** have и need для подсчета частот
2. **Подсчитываем уникальные числа** в паттерне (different)
3. **Реализуем функции** add **и** remove для обновления состояния окна, которые увеличивают и уменьшают счетчик уникальных значений (equal) в скользящем окне
4. **Проверяем** все возможные позиции окна длиной n в тексте
5. **Проверка совпадения** - когда equal == different, нашли решение

Python

```
MAX_NUMBER = 100000


def main():
    import sys
    input = sys.stdin.read
    data = input().split()

    idx = 0
```

```
n = int(data[idx]); idx += 1
pattern = []
for i in range(n):
    pattern.append(int(data[idx])); idx += 1

m = int(data[idx]); idx += 1
text = []
for i in range(m):
    text.append(int(data[idx])); idx += 1

have = [0] * (MAX_NUMBER + 1)
need = [0] * (MAX_NUMBER + 1)

# Count frequencies in pattern
different = 0
for num in pattern:
    need[num] += 1
    if need[num] == 1:
        different += 1

equal = 0

def add(x):
    nonlocal equal
    if have[x] == need[x] and need[x] != 0:
        equal -= 1
    have[x] += 1
    if have[x] == need[x]:
        equal += 1

def remove(x):
    nonlocal equal
```

```
        if have[x] == need[x]:
            equal -= 1
        have[x] -= 1
        if have[x] == need[x] and need[x] != 0:
            equal += 1

    # Initialize first window
    for i in range(n):
        add(text[i])

    if equal == different:
        print("YES")
        return

    # Slide the window
    for i in range(1, m - n + 1):
        add(text[i + n - 1])
        remove(text[i - 1])
        if equal == different:
            print("YES")
            return

    print("NO")

if __name__ == "__main__":
    main()
```

Уборка территории

Задачу можно решить методом двоичного поиска по ответу. Научимся проверять, можно ли разбить отрезок на n частей так, чтобы длины всех частей не превосходили x. Тогда ответом будет минимальное x, при котором будет существовать разбиение. Чтобы проверить существование разбиения при выбранном x, нужно действовать жадно.

Начало самого левого отрезка имеет координату 0, а концом сделаем точку с координатой $\min(a_2, x)$. Заметим, что конец точно не может иметь большую координату. Потому что тогда либо точки a_1 и a_2 обе будут лежать целиком внутри первого отрезка, либо первый отрезок будет иметь длину больше x. Также, несложно заметить, что уменьшать координату конца не имеет смысла, потому что если существует разбиение при котором конец первого отрезка имеет меньшую координату, эту координату можно увеличить.

При этом, длина первого отрезка не станет больше x, длина второго отрезка уменьшится, длины остальных отрезков не изменятся. Поэтому это разбиение тоже будет корректно. Таким образом, можно продолжать жадный алгоритм и для следующих отрезок, рассматривая в качестве координаты начала конец предыдущего отрезка. Если какой-то отрезок не содержит ни одной точки, либо конец последнего отрезка имеет координату меньше, чем l, разбиения при данном x не существует.

Решение на C++

```
#include <cmath>
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

typedef long long ll;

int main() {

 ios::sync_with_stdio(0);
 cin.tie(0);
 int n;
 ll len;
 cin >> n >> len;
 vector <ll> a(n);
 for (ll &x : a) cin >> x;
 sort(a.begin(), a.end());
 auto f = [&] (int t) {
  ll l = 0;
  for (int i = 0; i < n; i++) {
   if (a[i] - l > t) {
    return false;
```

```
    }
    ll nxt = (i == n - 1 ? len : a[i + 1]);
    l = min(nxt, l + t);
   }
   if (l < len) {
    return false;
   }
   return true;
  };
  ll l = 0, r = len;
  int it = 100;
  while (r - l > 1) {
   ll m = (l + r) / 2;
   if (f(m)) {
    r = m;
   } else {
    l = m;
   }
  }
  cout << r << endl;
}
```

Кодовый замок

Эта задача может быть решена методом динамического программирования. Вычислим значение dp[i][j] — количество последовательностей длины i, заканчивающихся на символ j, и соответствующих первым i числам из подсказки. Понятно, что dp[1][a_1] = 1, а все остальные dp[1][i] = 0. Для того, чтобы посчитать значение dp[i][j], нужно сложить dp[i − 1][j − a_i] и dp[i − 1][j + a_i] (каждое из значений прибавляется только в том случае, если существует соответствующее состояние).

Решение на C++

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

```
#include <map>
using namespace std;

vector<long long> dp1;
vector<long long> dp2;
vector<int> s;
int n;

const long long INF = 1000000007;

int main() {
    ios_base::sync_with_stdio(0);
    cin >> n;
    dp1.resize(26);
    dp2.resize(26);
    s.resize(n);
    for (int i = 0; i < n; i++) {
        cin >> s[i];
    }
    dp1[s[0]] = 1;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < 26; j++) {
            if (j + s[i + 1] < 26) {
                dp2[j + s[i + 1]] = (dp2[j + s[i + 1]] + dp1[j]) % INF;
            }
            if (j - s[i + 1] >= 0 && s[i + 1] != 0) {
                dp2[j - s[i + 1]] = (dp2[j - s[i + 1]] + dp1[j]) % INF;
            }
        }
        swap(dp1, dp2);
        fill(dp2.begin(), dp2.end(), 0);
    }
```

```
    long long ans = 0;
    for (int i = 0; i < 26; i++) {
        ans = (ans + dp1[i]) % INF;
    }
    cout << ans << endl;
    return 0;
}
```