|  |  |  |
| --- | --- | --- |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное автономное  образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Институт компьютерных наук и технологий | |
| **ОТЧЁТ**  по индивидуальной работе №2  по дисциплине «Язык программирования Python»  Вариант 12 | | |
|  | | Работу выполнил:  студент группы ИТ/О, ИТ-6-2025 НБ  1 курса Рисков Г. С.  «18» июня 2025 г. |
| Работу проверил:  Рубцова М.Б.  «19» июня 2025 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc201197703)

[Алгоритм решения 3](#_Toc201197704)

[Тестирование 5](#_Toc201197705)

[Код программы 6](#_Toc201197706)

# Постановка задачи

Шарики. В одной компьютерной игре игрок выставляет в линию шарики разных цветов. Когда образуется непрерывная цепочка из трех и более шариков одного цвета, она удаляется из линии. Все шарики при этом сдвигаются друг к другу, и ситуация может повториться. Напишите программу, которая по данной ситуации определяет, сколько шариков будет сейчас уничтожено. Естественно, непрерывных цепочек из трех и более одноцветных шаров в начальный момент может быть не более одной. Входные данные: даны количество шариков в цепочке (не более 105) и цвета шариков (от 0 до 9, каждому цвету соответствует свое целое число). Выходные данные: требуется вывести количество шариков, которое будет уничтожено.

Примеры:

(входные данные)

5 1 3 3 3 2

(выходные данные)

3

(входные данные)

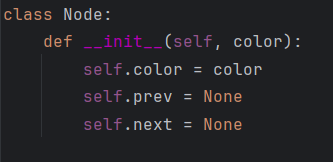
10 3 3 2 1 1 1 2 2 3 3

(выходные данные)

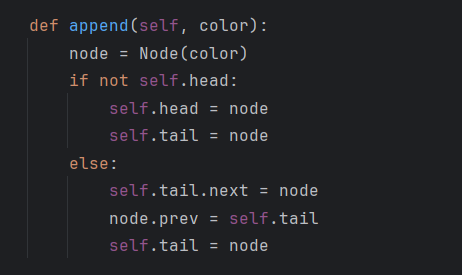
10

# Алгоритм решения

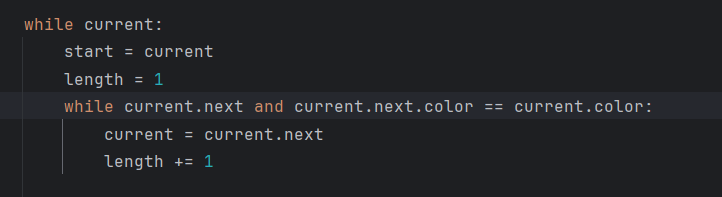
1. Для эффективного управления элементами используется двусвязный список



1. Список создаётся на основе входных данных. Он имеет указатели head и tail.

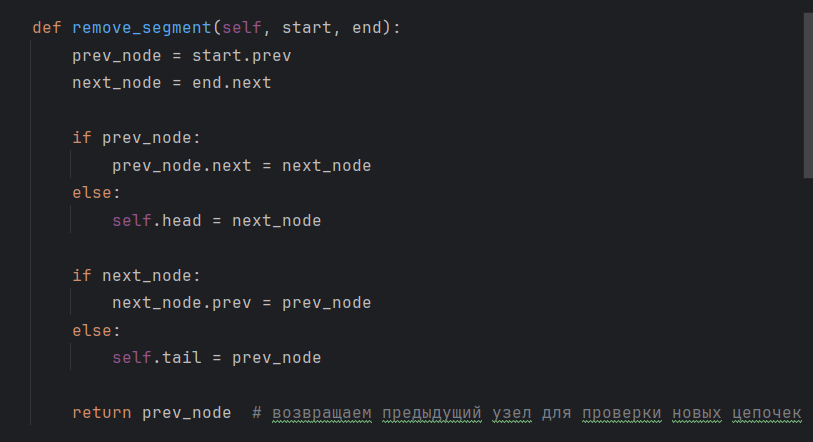


1. Поиск групп шариков
   * Программа проходит по списку с помощью указателя current
   * Для каждого узла определяется длина цепочки одинаковых цветов

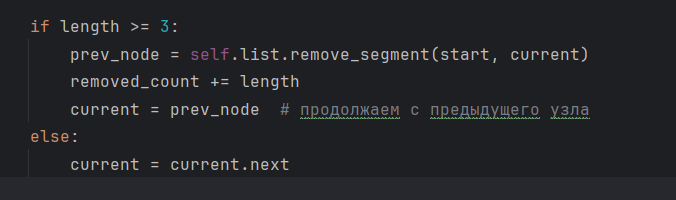


1. Удаление группы

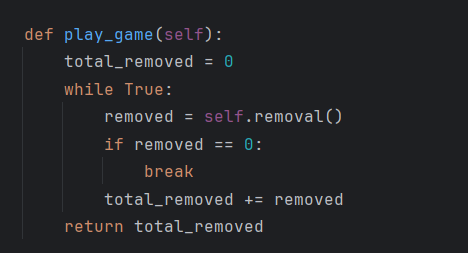
* Если группа содержит не менее трёх элементов — она удаляется, и обновляются связи.



1. После удаления программа возвращается к предыдущему узлу, чтобы проверить, не образовалась ли новая группа.



1. Метод play\_game() вызывает removal() до тех пор, пока есть что удалять.



# Тестирование

|  |  |  |
| --- | --- | --- |
| Входные данные | Результат | Примечание |
| 8 1 1 2 2 2 1 9 0 | Уничтожено шариков: 6 | Удаляются 2 2 2, затем 1 1 1 |
| 5 3 3 3 3 1 | Уничтожено шариков: 4 | Удаляется группа из 3+ подряд идущих цветов |
| 3 12 3 4 | Ошибка: цвет должен быть целым числом от 0 до 9. | Цвет вне диапазона |
| 5 1 2 3 | Ошибка: должно быть указано ровно 5 цветов, получено 3. | Должно быть указано n = 5 цветов |
|  | Ошибка: не введено ни одного числа. | Пользователь ничего не ввёл |
| 5 1 2 3 4 4.6 | Ошибка: введите только целые числа. | Цвета должны быть целыми числами |
| 5 | Ошибка: должно быть указано ровно 5 цветов, получено 0. | Нет значений цветов |

# Код программы

class Node:  
 def \_\_init\_\_(self, color):  
 self.color = color  
 self.prev = None  
 self.next = None  
  
  
class DoublyLinkedList:  
 def \_\_init\_\_(self, colors):  
 self.head = None  
 self.tail = None  
  
 for color in colors:  
 self.append(color)  
  
 def append(self, color):  
 node = Node(color)  
 if not self.head:  
 self.head = node  
 self.tail = node  
 else:  
 self.tail.next = node  
 node.prev = self.tail  
 self.tail = node  
  
 def remove\_segment(self, start, end):  
 prev\_node = start.prev  
 next\_node = end.next  
  
 if prev\_node:  
 prev\_node.next = next\_node  
 else:  
 self.head = next\_node  
  
 if next\_node:  
 next\_node.prev = prev\_node  
 else:  
 self.tail = prev\_node  
  
 return prev\_node # возвращаем предыдущий узел для проверки новых цепочек  
  
  
class BallGame:  
 def \_\_init\_\_(self, colors):  
 self.list = DoublyLinkedList(colors)  
  
 def removal(self):  
 removed\_count = 0  
 current = self.list.head  
  
 while current:  
 start = current  
 length = 1  
 while current.next and current.next.color == current.color:  
 current = current.next  
 length += 1  
  
 if length >= 3:  
 prev\_node = self.list.remove\_segment(start, current)  
 removed\_count += length  
 current = prev\_node # продолжаем с предыдущего узла  
 else:  
 current = current.next  
  
 return removed\_count  
  
 def play\_game(self):  
 total\_removed = 0  
 while True:  
 removed = self.removal()  
 if removed == 0:  
 break  
 total\_removed += removed  
 return total\_removed  
  
  
def main():  
 print(" Добро пожаловать в игру:")  
 print(" \"Шарики\"")  
 print()  
  
 colors = []  
  
 while True:  
 line = input("Введите последовательность: <n> <цвет1> <цвет2> ... (цвета от 0 до 9): ").strip()  
 try:  
 data = list(map(int, line.split()))  
 except ValueError:  
 print("Ошибка: введите только целые числа.")  
 print("Попробуйте ещё раз")  
 continue  
  
 if len(data) < 1:  
 print("Ошибка: не введено ни одного числа.")  
 print("Попробуйте ещё раз")  
 continue  
  
 n = data[0]  
 colors = data[1:]  
  
 if len(colors) != n:  
 print(f"Ошибка: должно быть указано ровно {n} цветов, получено {len(colors)}.")  
 print("Попробуйте ещё раз")  
 continue  
  
 if any(not (0 <= c <= 9) for c in colors):  
 print("Ошибка: цвет должен быть целым числом от 0 до 9.")  
 print("Попробуйте ещё раз")  
 continue  
  
 break  
  
 game = BallGame(colors)  
 destroyed = game.play\_game()  
  
 print(f"\nУничтожено шариков: {destroyed}")  
 print("Спасибо за игру!")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()