درس رایانش تکاملی - تمرین شماره یک (DarBabies)

نوید حاجی زاده – 4012366057

شرح مسئله :

براساس مثال DarBabies می‌خواهیم یک الگوریتم تکاملی بنویسیم و پارامترهای آن را به دلخواه انتخاب کنیم. در این مسئله رنگ خر داربی‌بی دارای هشت بیت است پس همین ابتدای امر می‌دانیم که اعداد ما بین 0 تا 255 خواهد بود همچنین پس‌زمینه را می‌توان سیاه یا سفید در نظر گرفت که در این مثال سیاه نماد عدد 0 و سفید نماد عدد 255 است. در هر نسل 16 خر داربی‌بی داریم که یک چهارم بدتر و یک چهارم تصادفی حذف می‌شوند و نسل جدید از نسل قدیم هر یک با یک بیت جهش بدست می‌آیند به گونه‌ای که از هر کدام از 8 خر داربی‌بی باقی مانده، 2 فرزند زاده شده و در نهایت نسل جدید مجددا 16 عدد خواهد داشت. در پایان میانگین اعداد یک نسل را محاسبه کرده و رسم می‌کنیم.

باید به این نکته توجه داشت که چون جهش‌ها تصادفی هستند نتایج هر اجرا با اجرای دیگر کمی متفاوت است ولی حالتِ کلی قابل تحلیل است. این الگوریتم با زبان برنامه نویسی پایتون نوشته شده است.

برای درک بهتر از الگوریتم گام به گام زیر استفاده می‌کنیم :

1- ابتدا بین 0 تا 255 ، 16 عدد تصادفی انتخاب کرده.
2- آنها را باینری می‌کنیم (8 بیت).
3- پس‌زمینه را 0 در نظر می‌گیریم.
4- 4 عدد که بیشترین اختلاف را با پس زمینه دارند حذف می‌کنیم.
5- 4 عدد را نیز به صورت تصادفی حذف کرده.

6- از 8 عدد باقی مانده ، از هر عدد، 2 عدد تازه برای نسل بعد می‌سازیم به گونه‌ای که یک بیت به صورت تصادفی تغییر کند.

7- میانگین اعداد در یک نسل را محاسبه کرده و در یک لیست ذخیره سازی می‌کنیم.

8- حال الگوریتم بالا را برای چند نسل تکرار کرده.

9- در نهایت نمودار میانگین اعداد هر نسل را که در لیست ذخیره کرده بودیم، رسم می‌کنیم.

با توجه به شرح مسئله و الگوریتم بیان شده، از آنجا که سعی کردیم تا کمی نظریه تکامل را شبیه سازی کنیم و از دو بازوی اصلی آن یعنی "انتخاب طبیعی" و "جهش" بهره ببریم، لذا انتظار داریم تا بعد از چند نسل، اعداد تصادفی که در نسل اول تولید کرده بودیم، به رنگ پس‌زمینه تا حدودی همگرا شوند (یا به عبارت بهتر، تنها فرزندانی باقی بمانند که می‌توانند خود را با محیط اطراف سازگارتر کنند و شکار نشوند). البته ذکر این نکته بسیار مهم است که اگر به خوبی در الگوریتم بالا توجه کنیم، متوجه می‌شویم که میزان تصادفی بودن در این الگوریتم بسیار بالاست. پس باید این احتمال را بدهیم که در برخی از نسل‌ها میانگین اعداد یک نسل به پس‌زمینه نه تنها نزدیک نشود بلکه دور هم بشود.

در نهایت با توجه به کد پایتونی که در بخش پیوست تقدیم شده است، خروجی ما به صورت زیر می‌باشد :

با توجه به مطالب بیان شده، مشاهده می‌شود که خروجی آنچه را انتظار داشتیم، نمایش می‌دهد.

در پیوست زیر کد مربوط به خروجی بالا قرار داده شده است.

پیوست :

```
import random
import matplotlib.pyplot as plt

def generate_binary_list(num_elements):
    Bin_list = []
    for _ in range(num_elements):
        random_num = random.randint(0, 255)
        binary_num = format(random_num, '08b')
        Bin_list.append(binary_num)
    return Bin_list

def find_and_remove_elements(input_list, num_to_remove=4):
    max_indices = sorted(range(len(input_list)), key=lambda x: input_list[x],
reverse=True)[:num_to_remove]

    for index in sorted(max_indices, reverse=True):
        input_list.pop(index)

    for _ in range(num_to_remove):
        index = random.randrange(len(input_list))
        input_list.pop(index)
    return Bin_list


def flip_bits(bin_list, num_flips=2, flip_probability=0.5):
    new_Bin_list = []
    for i in range(len(bin_list)):
        for _ in range(num_flips):
            if random.random() < flip_probability:
                index = random.randint(0, 7)
                number = list(bin_list[i])
                bit = number[index]
                number[index] = "0" if bit == "1" else "1"
                new_Bin_list.append("".join(number))
            else:
                new_Bin_list.append(bin_list[i])  # Keep the bit unchanged
    return new_Bin_list

# Initial generation
num_elements = 16
```

```python
num_to_remove = 4
num_flips = 2
Bin_list = generate_binary_list(num_elements)
print('fisrt gen is :',Bin_list)

# Iteration time
results = []
iterations = []
itration_counter = 1

for iteration in range(5):
    iterations.append(iteration)
    print("Iteration:", itration_counter)
    #print("Initial Bin_list:", Bin_list)

    # Find and remove elements
    find_and_remove_elements(Bin_list)
    #print("After removing 4 max and 4 random elements:", Bin_list)

    # Flip bits
    Bin_list = flip_bits(Bin_list)
    #print("After flipping bits:", Bin_list)
    result = sum(int(bin_str, 2)/16 for bin_str in Bin_list)
    results.append(result)
    #print("Sum of Bin_list is :", result)
    itration_counter += 1
    print(itration_counter ,' gen is :',Bin_list)

# Create the graph
plt.plot(iterations, results)
plt.xlabel("Iterations")
plt.ylabel("Result Value")
plt.title("Result Value vs. Iterations")
plt.grid(True)
plt.show()
plt.savefig("result_plot.png")
```