به نام خدا

تمرین سری ششم

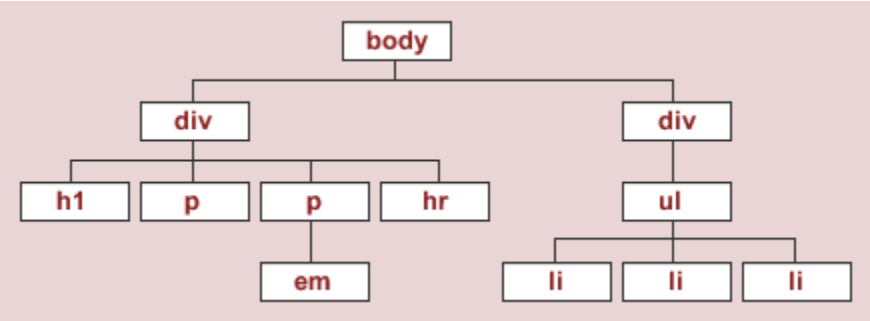
درس مبانی هوش محاسباتی

سینا علی‌نژاد

۹۹۵۲۱۴۶۹

سوال ۱-

الف) ساختار ژنوم: برای ساختار ژنوم میتوان از همان parse tree مربوط به html استفاده کرد. Node ها همان تگها مثل html, body, head, p, a, ul خواهند بود و یالهای بین اینها رابطه والد و فرزندی را مشخص میکند. یعنی اگر یک گره فرزند گره دیگری بود، یعنی یک تگ درون یک تگ دیگر قرار گرفته. شکل زیر یک درخت با این ویژگیها را نشان میدهد.



جمعیت اولیه را با استفاده از روش ramped half-half میسازیم و ماکزیمم عمق را برابر با ۱۰ میگیرم.

ب) تابع برازندگی(fitness function): برای این کار میتوان از فاکتورهای زیر استفاده کرد:

- **Semantic Accuracy:** Does the design match the content and purpose specified by the user?
- **Visual Balance & Appeal:** Is the layout pleasing to the eye, with appropriate use of space, color, and hierarchy?
- **User Interaction & Usability:** Is the design intuitive and easy to navigate for the target audience?
- **Technical Feasibility:** Can the design be implemented using standard web technologies like HTML, CSS, and JavaScript?

F(tree) = w1 * SemanticAccuracy(tree) + w2 * VisualAppeal(tree) + w3 * UserInteraction(tree) + w4 * TechnicalFeasibility(tree)

از طرفی میتوان با استفاده تکنیکهای text mining یک سری کلیدواژه از متن درخواستی کاربر استخراج کرد و بررسی کرد که ژنوم ساخته شده چقدر با این کلیدواژه ها overlap دارد و سپس بر اساس آن تابع برازندگی را تعریف کرد.

ج) با استفاده از توضیحات گفته شده توسط کاربر میتوان کلیدواژه های زیر را با استفاده از روشهای مختلف text mining استخراج کرد.

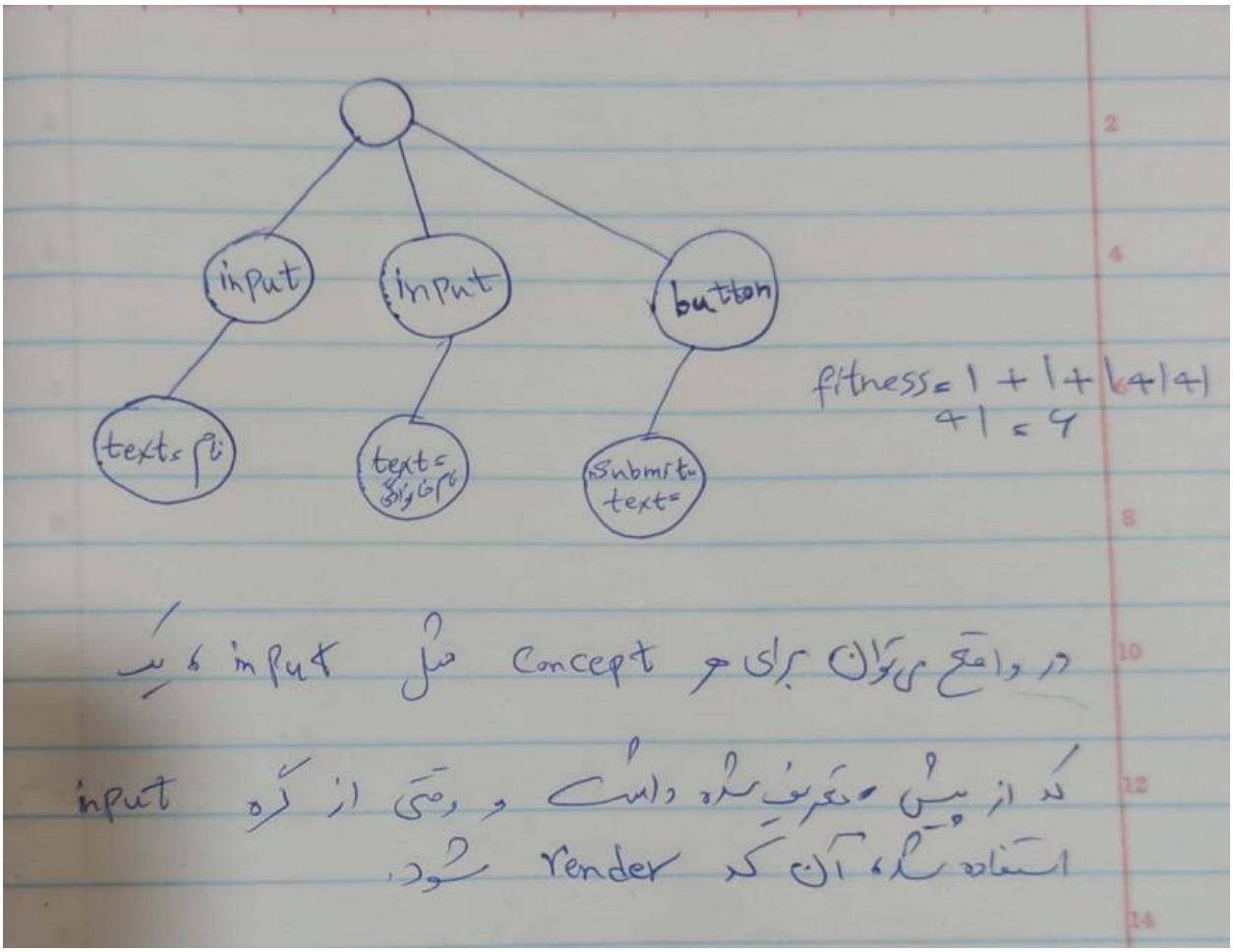
- input for name

- input for lastname

- button in the end

در عکس زیر، فرض کردم پس از چند مرحله به چنین چیزی میرسیم. مراحل میانی را در نظر نگرفتم.

در واقع باید mutation و crossover روی جمعیت اعمال شود و پس از آن به وسیله تابع fitness فرایند انتخاب صورت بگیرد تا پس از چندین نسل به درختهایی با fitness بالا برسیم.



سوال ۲- برای این سوال از کتابخانه deap برای اجرای الگوریتم ژنتیک استفاده کردم.

کتابخانه های مورد نیاز:

```
import operator
import random
import numpy as np
from deap import algorithms, base, creator, tools
```
✓ 0.5s

تابعی که میخواهیم ریشه آن را بدست بیاوریم:

```
# Define the polynomial
def polynomial(x):
    return 4 * x**3 - 5 * x**2 + x - 1
```

تابع fitness نیز باید تعریف شود تا مناسب بودن هر عضو از جمعیت را با استفاده از آن تعیین کنیم.

```
# Define the fitness function
def fitness_func(individual):
    x = individual[0]
    return (np.abs(polynomial(x)),)
```

حال باید به وسیله ماژول base یک toolbox تعریف کنیم تا چگونگی ساخت جمعیت اولیه، mutaion، crossover و selection را مشخص کنیم.

```
toolbox.register("attr_float", random.uniform, -100, 100)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_float, n=1)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
```

در سه خط بالا، مشخص میکنیم که جمعیت اولیه باید به صورت یکنواخت بین 100- و 100 پخش شده باشند. پس فضای جستجوی ما بین این دو عدد است.

```
toolbox.register("evaluate", fitness_func)
toolbox.register("mate", tools.cxBlend, alpha=0.5)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=10, indpb=0.1)
toolbox.register("select", tools.selTournament, tournsize=3)
```

در خطوط بالا به ترتیب، تابع ارزیابی یا برازندگی، روش crossover، mutation و selection تعیین میشود.

عملگرهای ژنتیکی مورد استفاده در این کد عبارتند از:

selection: انتخاب tournament با سایز 3.

crossover: blend crossover با مقدار آلفا 0.5

mutation: gaussian mutation با میانگین 0، انحراف معیار 10 و احتمال 0.1

```
# Create the population
population = toolbox.population(n=50)
```

ساخت جمعیت اولیه با تعداد ۵۰ تا.

```
# Run the genetic algorithm
for gen in range(100):
    offspring = algorithms.varAnd(population, toolbox, cxpb=0.5, mutpb=0.1)
    fits = toolbox.map(toolbox.evaluate, offspring)
    for fit, ind in zip(fits, offspring):
        ind.fitness.values = fit
    population = toolbox.select(offspring, k=len(population))
```

اجرای الگوریتم برای ۱۰۰ نسل و سپس توقف. در هر بار اجرا، با استفاده از mutation و crossover به جمعیت افزوده میشود و در آخر از بین کل جمعیت selection انجام میشود. احتمال mutation همان 0.1 و احتمال crossover برابر با 0.5 گرفتیم.

در آخر سه تا از بهترین جوابها را چاپ کردیم:

```
# Print the best solutions
best_individual = tools.selBest(population, k=3)
for i, individual in enumerate(best_individual):
    print("The best solution for example {i} is: x = {solution} with fitness = {fitness}".format(i=i+1, solution=individual[0],
    fitness=individual.fitness.values[0]))
```

جواب برای تابع تعیین شده در اول گزارش:

```
The best solution for example 1 is: x = 1.2137289642203735 with fitness = 0.0
The best solution for example 2 is: x = 1.2137289642203735 with fitness = 0.0
The best solution for example 3 is: x = 1.2137289642203735 with fitness = 0.0
```

جواب برای تابع زیر

- $186x^3-7.22x^2+15.5x-13.2=0$

به صورت:

```
The best solution for example 1 is: x = 0.34897658886936966 with fitness = 0.7651490304624478
The best solution for example 2 is: x = 0.34897658886936966 with fitness = 0.7651490304624478
The best solution for example 3 is: x = 0.34897658886936966 with fitness = 0.7651490304624478
```

جواب برای تابع زیر:

- $4x^3-5x^2+x-1=0$

به صورت:

```
The best solution for example 1 is: x = 1.2137289642203735 with fitness = 0.0
The best solution for example 2 is: x = 1.2137289642203735 with fitness = 0.0
The best solution for example 3 is: x = 1.2137289642203735 with fitness = 0.0
```

جواب برای تابع زیر:

- $X^2-8x+4=0$

به صورت:

```
The best solution for example 1 is: x = 7.464307036542161 with fitness = 0.0014232434355321288
The best solution for example 2 is: x = 7.464307036542161 with fitness = 0.0014232434355321288
The best solution for example 3 is: x = 7.464307036542161 with fitness = 0.0014232434355321288
```

جواب برای تابع زیر:

- $2x-4=0$

به صورت:

```
The best solution for example 1 is: x = 2.0 with fitness = 0.0
The best solution for example 2 is: x = 2.0 with fitness = 0.0
The best solution for example 3 is: x = 2.0 with fitness = 0.0
```

سوال ۳-

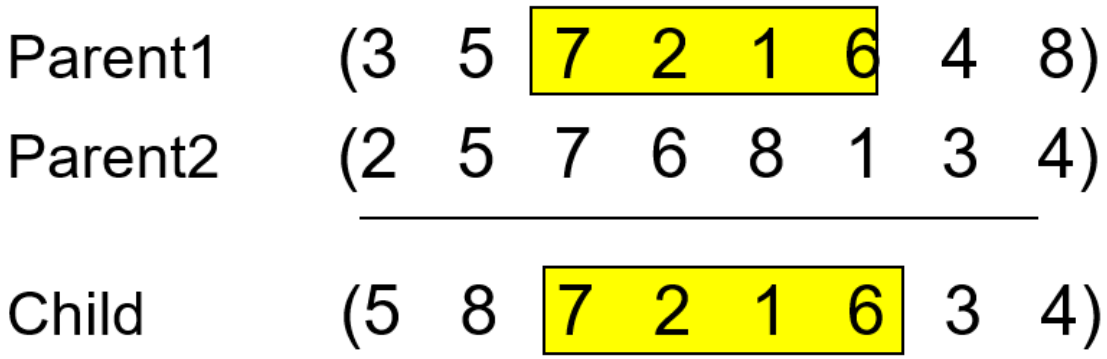
سوال ۳ (۲۰ نمره)

فرض کنید یک مربع ۶×۶ داریم که می خواهیم اعدادی بین ۱ تا ۳۶ را در این مربع قرار دهیم به گونه‌ای که تعداد اعداد زوج و فرد در هر سطر و ستون برابر باشند. این مسئله را به روش الگوریتم ژنتیک حل نمایید. ساختار ژنوم، تابع برازندگی (*fitness function*) و کلیه پارامترهای لازم برای حل مسئله را تعریف نمایید.

ساختار ژنوم: ژنوم ما در اینجا به صورت یک ماتریس 6*6 میباشد و در هر خانه یک عدد بین ۱ تا ۳۶ حضور دارد.

First population: تعداد جمعیت را ۵۰ میگیرم و به این تعداد ماتریس 6*6 با اعداد رندوم بین ۱ تا ۳۶ میسازم. فقط باید توجه شود که این ماتریس عدد تکراری نمیتواند داشته باشد و همه اعداد ۱ تا ۳۶ باید حضور داشته باشند.

Mutation and Crossover: ماتریس را میتوان flat کرد و در حالت flat شده Crossover را انجام داد. برای این کار از مشابهش در مسئله TSP استفاده میکنیم. یعنی یک قسمت از یکی را مستقیما در فرزند کپی کرده و بقیه را از والد دیگر به گونه ای انتخاب میکنیم که اعداد تکراری نداشته باشیم. مشابه شکل زیر:



تابع برازندگی(Fitness function):

$$f = -\sum_{\text{سطر و ستون}} abs(num\_of\_1s - num\_of\_0s)$$

هرچه عدد بالا به صفر نزدیک تر شود، تابع برازندگی بهتری است. اگر به صفر برسد یعنی به بهترین جواب رسیده‌ایم.

Selection: برای انتخاب، از روش baker method که از Roulette wheel بهره میبرد، استفاده میکنیم.

Termination: بعد از تعداد مشخصی نسل، الگوریتم را متوقف میکنیم. برای مثال پس از ۵۰ نسل.

سوال ۴-

9 % 3 + 1 = 1

پس حالت اول باید شناسایی شود.

با استفاده از روش PSO این مسئله را حل میکنم.

برای این مسئله تعداد particle ها را برابر با n=100 میگیرم. برای هر کدام از ۵ عکس ورودی، یک نقطه در فضای یک بعدی و به صورت افقی در کنار هم در نظر میگیرم و particle ها را در این نقاط که تعدادشان ۵ تاست پخش میکنیم. از آنجا که فضا یک‌بعدی است، برای هر particle یک x و یک v در نظر میگیریم.

$$X = [x_1, x_2, \dots, x_n]$$
$$V = [v_1, v_2, \dots, v_n]$$

تابع برازندگی یا fitness: هر عکس را متناظر یک نقطه گرفتیم. هر مورچه که در نقطه متناظر با یکی از این عکسها قرار گرفت، با استفاده از correlation بین عکس اصلی (حالت اول) با عکس فعلی، میتواند fitness را برای خودش حساب کند. برای مثال اگر دو خانه متناظر از این دو عکس، یکی پیکسل روشن و دیگری پیکسل خاموش داشت، از مقدار fitness یکی کم و اگر یکسان بودند، به تابع fitness یکی اضافه میکنیم. پس بیشترین مقدار fitness عدد 16 و کمترین آن عدد 16- خواهد بود که برای زمانی است که دو عکس کاملا منفی یکدیگرند.

سرعت هر particle را با فرمول زیر بروز میکنیم.

$$\mathbf{v}_i(k+1) = \omega \times \mathbf{v}_i(k) + c_1 \times random_1() \times (PBest_i - \mathbf{x}_i(k)) + c_2 \times random_2() \times (GBest - \mathbf{x}_i(k))$$

مقادیر ثابت را به صورت زیر انتخاب میکنیم.

$$c_1 = 2, c_2 = 2, w = 0.9$$

که طبق اسلایدها مقادیر مناسبی برای این الگوریتم هستند.

برای بروزرسانی جایگاه هر particle از فرمول زیر استفاده میکنیم:

$$\mathbf{x}_i(k+1) = \mathbf{x}_i(k) + \mathbf{v}_i(k+1)$$

مقادیر Gbest و Pbest نیز باید هربار بروز شوند. هر particle یک Pbest مخصوص خودش دارد و از بین این Pbest ها یکی که بهترین است به عنوان Gbest نگه داشته میشود.

Termination: مسئله را تا ۲۰۰ iteration پیش می‌بریم و در نهایت با توجه به تابع برازندگی تعریف شده انتظار می‌رود مورچه ها یا همان particleها روی عکسهایی که همانند حالت اول هستند، جمع شده باشند. هرچه بیشتر شبیه حالت اول باشند، مورچه های بیشتری روی آنها خواهد بود.