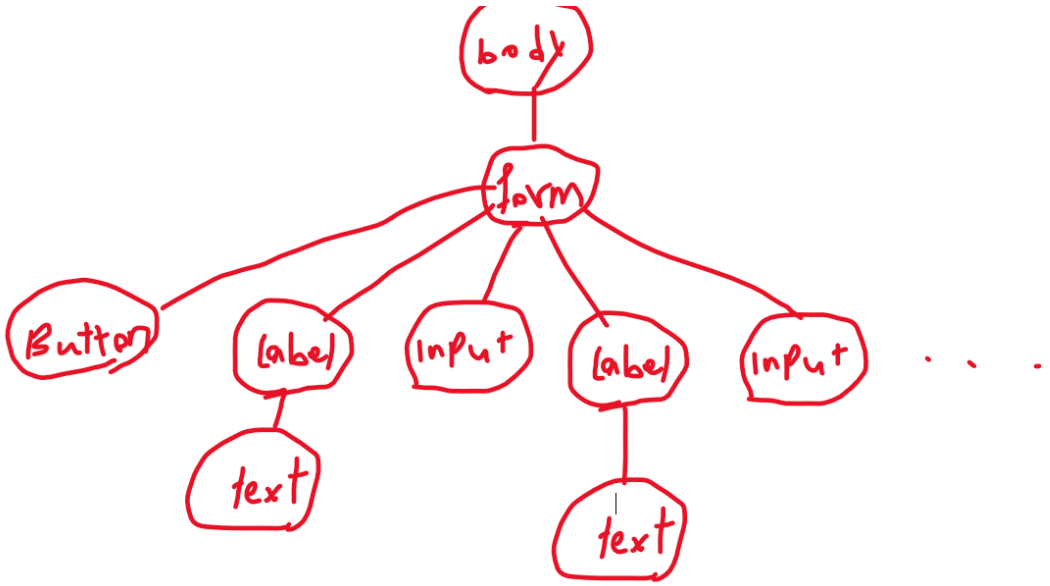
(Q1

روند کلی و ساختار درخت:

الگوریتم GP یا برنامه‌نویسی ژنتیکی، یک روش محاسباتی است که بر اساس ایده‌های اصلی از تکامل و انتخاب طبیعی برای حل مسائل بهینه‌سازی و مسائل پیچیده استفاده می‌کند. در این مسئله، از درخت DOM در html به عنوان نمونه در GP استفاده میشود. هر درخت نشان دهنده یک صفحه وب طراحی شده است. برای مثال صفحه html زیر به این صورت به یک درخت تبدیل میشود.

```
<body>

<form>
  <label for="name">نام:</label>
  <input type="text" id="name" name="name" required>

  <label for="email">ایمیل:</label>
  <input type="email" id="email" name="email" required>

  <label for="message">پیام:</label>
  <textarea id="message" name="message" rows="4" required></textarea>

  <button type="submit">ارسال</button>
</form>

</body>
```



برای استفاده از این درخت در این الگوریتم این مراحل را انجام میدهیم:

۱. **تولید جمعیت اولیه :(Initialization)** در این مرحله، یک جمعیت اولیه درخت های DOM مختلف ایجاد می‌شود. این جمعیت اولیه به صورت تصادفی ایجاد می‌شود و شامل تگ ها و عناصری است که از پیش تعریف شده.

۲. **ارزیابی :(Evaluation)** در این مرحله، هر نمونه در جمعیت اولیه بر اساس یک تابع هدف یا تابع ارزیابی ارزیابی می‌شود. این تابع هدف معیاری است که برای اندازه‌گیری کیفیت هر نمونه استفاده می‌شود. این تابع در ادامه شرح داده شده.

۳. **انتخاب :(Selection)** مجموعه‌ای از نمونه ها بر اساس نتایج ارزیابی به عنوان والدین برای مرحله بعدی انتخاب می‌شوند. از روش رولت ویل برای selection در این قسمت استفاده میشود.

۴. **تولید نسل جدید :(Reproduction)** در این مرحله، ابزارهای انتخاب شده به عنوان والدین استفاده می‌شوند تا ابزارهای جدیدی تولید کنند. این کار به دو روش ممکن است. هم از طریق Mutation هم از طریق cross over. در mutation، یک قسمت از درخت با یک درخت تصادفی تولید شده جایگزین میشود و در روش cross over، یک نود از هر دو والد انتخاب شده و با هم جا به جا می شوند.

۵. **پایان الگوریتم یا مرحله تکرار** :مراحل ۲ تا ۴ به ترتیب تکرار می‌شوند تا شرایط پایانی تعیین شده برای الگوریتم به دست آید. این شرایط میتواند تعداد نسل‌های تولید شده یا دستیابی به یک مقدار معین از کیفیت باشد.

۶. **بررسی بهترین حل :(Best Solution)** در پایان الگوریتم، بهترین نمونه یا جمعیتی که بهترین عملکرد را در حل مسئله داشته‌اند، انتخاب و گزارش می‌شود.

۲. یک *fitness function* برای این امر تعریف نمایید. (توضیحات جهت علت انتخاب این *fitness function* الزامی است و طبیعتا به توابع با دقت و کارایی بالاتر نمره بیشتری تعلق می‌گیرد.)

برای این قسمت میتوانیم تابع fitness را به این صورت تعریف کنیم که ویژگی های مثبت و منفی هر نمونه را پیدا کنیم و به هر ویژگی یک وزن اختصاص دهیم. به ویژگی های مهم تر مثل functionality درخت، وزن بیشتری میدهیم و به ویژگی های کم اهمیت تر مثل زیبایی ظاهری وزن کم تری میدهیم. به معایب درخت هم وزن منفی اختصاص میدهیم.

ویژگی های درخت شامل مزایا و معایب میتوانند به این صورت باشند:

مزایا:

- Functionality درست صفحه
- زیبایی ظاهری صفحه
- کوچک بودن درخت DOM
- مطابق بودن با توضیحات کاربر
- وجود تگ های ضروری در درخت مثل form و submit

معایب:

- حجیم شدن درخت DOM
- عدم تنوع در استفاده از تگ های html
- خطای اجرای صفحه html از روی درخت

اگر برای هر کدام از این موارد یک امتیاز را حساب کنیم در نهایت تابع fitness به این صورت خواهد بود:

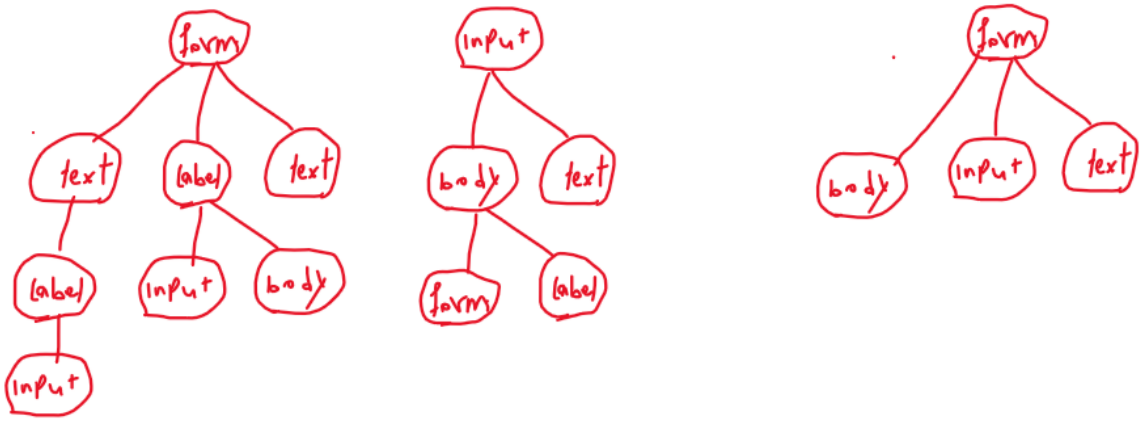
Fitness = w1.s1 + w2.s2 + w3.s3 + w4.s4 + ...

۳. این ابزار را روی مثال زیر به صورت دستی انجام دهید تا به جواب برسید.

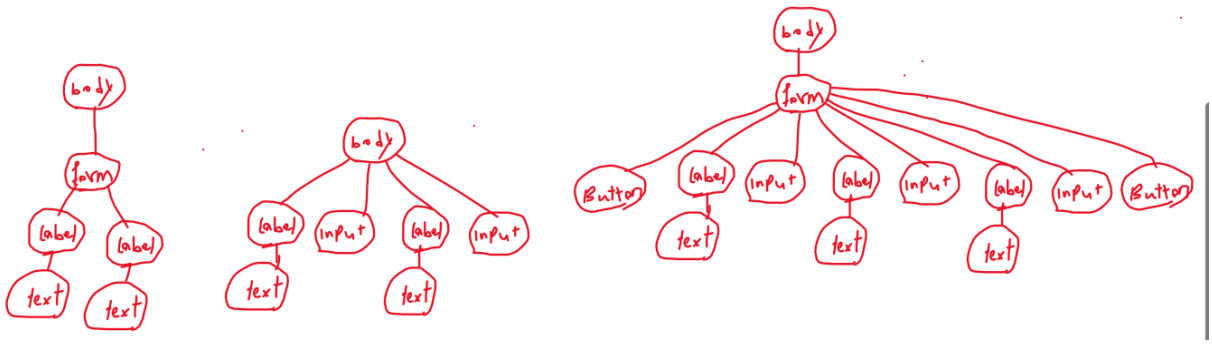
«یک فرم ثبت نام شامل ۲ عدد باکس ورودی که نام و نام خانوادگی را دریافت می‌کند. همچنین این فرم در انتهای خود یک دکمه جهت تکمیل ثبت نام دارد»

توضیحات کاربر عبارت است از: "یک فرم ثبت نام شامل ۲ عدد باکس ورودی که نام و نام خانوادگی را دریافت می کند. همچنین این فرم در انتهای خود یک دکمه جهت تکمیل ثبت نام دارد"

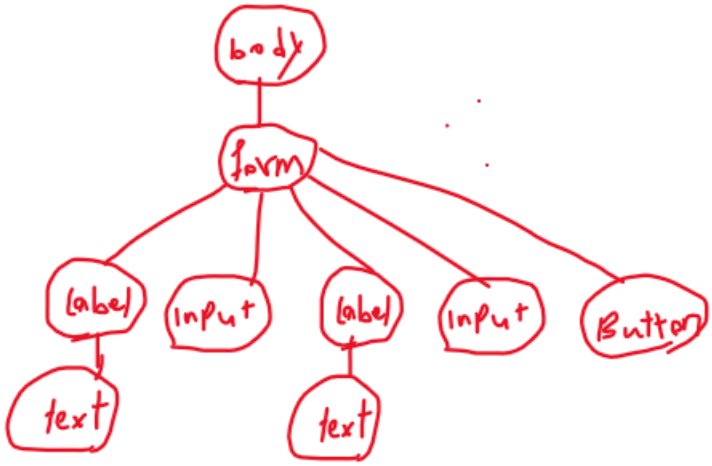
در ابتدا درخت های تصادفی تولید شده بی معنا هستند:



اما به مرور با تابع فیتنسی که تعریف کردیم بهتر میشوند و شکل یک صفحه وب بدون خطا میگیرند:



در نهایت مواردی که خطای اجرا دارند(وسطی) و یا درخت بسیار بزرگی دارند(راستی) و یا مطابق با درخواست کاربر نیستند(چپی) به مرور حذف شده و درخت نهایی احتمالا به این شکل خواهد بود:



Q2)

برای حل این سوال از الگوریتم genetic استفاده میکنم. ابتدا جزعیات الگوریتم را برای حل این مسئله توضیح میدهم، سپس کد را شرح میدهم و در نهایت نتایج را بررسی میکنم:

۱- Encoding:

برای حل این مسئله ابتدا باید شکل کروموزوم ها را مشخص کنیم. کروموزوم ها ها را به شکل یک آرایه ۲۰ تایی از ۰ و ۱ ایجاد میکنیم که ۳ خانه اول نشان دهنده توان های ۰ تا ۲ و سایر خانه ها اعشار را مشخص میکنند.(برای حل نمونه توابع مشخص شده در این سوال، این بازه کافی بود. در صورت لزوم میتوان بازه اعداد را بیشتر کرد یا تعداد خانه های آرایه را افزایش داد)

۲- Initialization:

برای شروع الگوریتم از جمعیت ۵۰ عضوی استفاده میکنیم. هر کروموزوم را به صورت رندوم با اعداد ۰ تا ۱ پر میکنیم.

۳- Fitness:

از قدر مطلق مقدار تابع در هر نقطه به عنوان معیار فیتنس آن نقطه استفاده میکنیم.

۴- Selection:

از روش رولت ویل برای انتخاب کردن اعضای نسل بعد استفاده میکنیم. به هر عضو، متناسب با فیتنس آن شانس انتخاب شدن میدهیم

۵- combination:

برای ترکیب کردن دو عضو، مقادیر خانه های آنها را به صورت رندوم جا به جا میکنیم.

۶- Mutation:

با احتمال p=0.001 یک خانه را تاگل میکنیم.(صفر را یک میکنیم و یک را صفر میکنیم)

۷- شرط پایان:

مراحل ۳ تا ۶ تکرار میشوند تا وقتی شرط پایان براورده شود. شرط پایان میتواند رسید به فیتنس دلخواه یا پیش رفتن تا حداکثر ۵۰ یا ۱۰۰ نسل باشد.

حالا کد را توضیح میدهم:

در این قسمت تابع هدف را مشخص کرده ام:

```
def target_function(x):
    return x**2 - 4
```

برای پیشبینی هر تابع کافی است این تابع را تغییر دهیم.

این تابع برای تبدیل کروموزوم به عدد دسیمال است:

```
def binary_to_decimal(num):
    result = 0
    for i in range(len(num)):
        result += 2 ** (3 - i - 1) * num[i]
    return result
```

در این قسمت متغیر های اصلی برنامه را تنظیم کردم:

```
population_size = 50
generations = 50
mutation_rate = 0.1
binary_length = 20
report_step = 10
```

```
population = np.random.choice([0, 1], size=(population_size, binary_length), replace=True)
```

```
for generation in range(generations+1):
    decimal_values = np.apply_along_axis(binary_to_decimal, 1, population)

    fitness_values = np.abs(target_function(decimal_values))

    selected_indices = np.argsort(fitness_values)[:int(population_size / 2)]

    crossover_indices = np.random.choice(selected_indices, size=int(population_size / 2), replace=True)
    offspring = np.vstack([population[selected_indices], population[crossover_indices]])

    mutation_mask = np.random.rand(population_size, binary_length) < mutation_rate
    offspring ^= mutation_mask

    population = offspring
```

این قسمت لوپ اصلی برنامه است که در آن نسل ها را طبق الگوریتم GA در اسلایدها جلو بردم.

```
if generation % report_step == 0:
    best_solution_binary = population[np.argmin(np.abs(target_function(np.apply_along_axis(bi
    gen_best_solution = binary_to_decimal(best_solution_binary)
    gev_best_value = target_function(gen_best_solution)
    print(f"=========== Gen {generation} ===========")
    print(f"Best solution:\t {gen_best_solution}")
    print(f"Function value:\t {gev_best_value}")
```

این قسمت هم برای لاگ انداختن حین اجرا و دیدن مراحل پیشرفت برنامه است.

بررسی نتایج:

```
def target_function(x):
    return 2*x - 4
```

```
========== Gen 0 ==========
Best solution:   2.1255340576171875
Function value:  0.251068115234375
========== Gen 20 ==========
Best solution:   2.005279541015625
Function value:  0.01055908203125
========== Gen 40 ==========
Best solution:   2.0045318603515625
Function value:  0.009063720703125
========== Gen 60 ==========
Best solution:   2.0077362060546875
Function value:  0.015472412109375
========== Gen 80 ==========
Best solution:   2.0034866333007812
Function value:  0.0069732666015625
========== Gen 100 ==========
Best solution:   2.0011749267578125
Function value:  0.002349853515625
```

میبنیم که خطا از ۰.۲۵ به ۰.۰۰۲ رسیده و الگوریتم به خوبی کار کرده.

```python
def target_function(x):
    return x**2 - 8*x + 4
```

```
========== Gen 0 ==========
Best solution:   0.47772216796875
Function value:  0.4064411260187626
========== Gen 20 ==========
Best solution:   0.5438690185546875
Function value:  -0.055158639093860984
========== Gen 40 ==========
Best solution:   0.5340728759765625
Function value:  0.01265082904137671
========== Gen 60 ==========
Best solution:   0.5364990234375
Function value:  -0.004160985350608826
========== Gen 80 ==========
Best solution:   0.53057861328125
Function value:  0.03688475862145424
========== Gen 100 ==========
Best solution:   0.5392227172851562
Function value:  -0.023020599444862455
```

مجددا الگوریتم به خوبی کار کرده.

```python
def target_function(x):
    return 4 * x **3 - 5 * x ** 2 + x - 1
```

```
========== Gen 0 ==========
Best solution:   1.0679473876953125
Function value:  -0.7625850935451552
========== Gen 20 ==========
Best solution:   1.2120513916015625
Function value:  -0.010945040785017568
========== Gen 40 ==========
Best solution:   1.216888427734375
Function value:  0.020759652291758357
========== Gen 60 ==========
Best solution:   1.2156448364257812
Function value:  0.012565642255127685
========== Gen 80 ==========
Best solution:   1.2132186889648438
Function value:  -0.0033348971603235356
========== Gen 100 ==========
Best solution:   1.2081069946289062
Function value:  -0.03646814285271738
```

برای این مورد هم به درستی اجرا میشود.

```python
def target_function(x):
    return 186 * x **3 - 7.22 * x ** 2 + 15.5 * x - 13.2
```

```
========== Gen 0 ==========
Best solution:   0.2542572021484375
Function value:  -6.668500178096286
========== Gen 20 ==========
Best solution:   0.3572845458984375
Function value:  -0.10061820841280067
========== Gen 40 ==========
Best solution:   0.35985565185546875
Function value:  0.110382221871102 36
========== Gen 60 ==========
Best solution:   0.358673095703125
Function value:  0.0130178278201071 86
========== Gen 80 ==========
Best solution:   0.36496734619140625
Function value:  0.53749946115623
========== Gen 100 ==========
Best solution:   0.35214996337890625
Function value:  -0.5144095171523535
```

برای این مورد هم الگوریتم خطا را تا حد زیادی کاهش داده اما به اندازه موارد قبلی دقیق نشده چون در این مورد، تابع شیب خیلی زیادی در ریشه دارد و کوچک ترین تغییرات x باعث تغییرات شدید مقدار تابع میشود. برای بهتر کردن دقت کافیست تعداد ارقام اعشار را افزایش داد.

برای حل این سوال از چت جی پی تی هم کمک گرفته شده.

Q3)

۱- Encoding:

برای حل این مسئله ابتدا باید شکل کروموزوم ها را مشخص کنیم. کروموزوم ها ها را به شکل یک جدول ۶ در ۶ انتخاب میکنیم که هر کدام یک جواب برای مسئله هستند.

۲- Initialization:

برای شروع الگوریتم از جمعیت ۱۰۰ عضو استفاده میکنیم. خانه های هر جدول به صورت رندوم و تصادفی از ۱ تا ۳۶ پر میشوند.

۳- Fitness:

از جمع اختلاف تعداد اعداد زوج و فرد در هر سطر و ستون میتوانیم به عنوان معیاری برای فیتنس استفاده کنیم. گزینه دیگر برای فیتنس میتواند تعداد سطر ها و ستون هایی که تعداد اعداد زوج و فرد آنها برابر است باشد.

۴- Selection:

از روش رولت ویل برای انتخاب کردن اعضای نسل بعد استفاده میکنیم. به هر عضو، متناسب با فیتنس آن شانس انتخاب شدن میدهیم

۵- Cross over:

برای ترکیب کردن دو عضو، با احتمال یک دوم، سطر دوم و سوم و با احتمال یک دوم هم ستون دوم و سوم دو کروموزوم والد را با هم جا به جا میکنیم.

۶- Mutation:

با احتمال p=0.001 جای دو خانه تصادفی از یک جدول را با هم عوض میکنیم

۷- شرط پایان:

مراحل ۳ تا ۶ تکرار میشوند تا وقتی شرط پایان براورده شود. شرط پایان میتواند رسید به فیتنس دلخواه یا پیش رفتن تا حداکثر ۱۰۰ نسل باشد.

Q4)

شماره دانشجویی: 99521073 => s = 1

برای حل این سوال از روش PSO استفاده میکنیم. تصویر مربوط به من، تصویر حالت اول است.

برای حل این مسئله به کمک PSO، ابتدا تصاویر را روی محور مختصات قرار میدهیم. هر تصویر نمایانگر یک نقطه روی محور مختصات میشود.

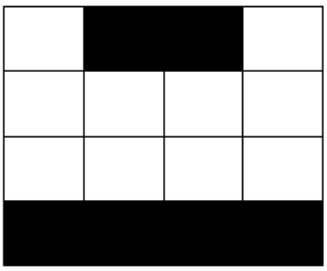
۱- Initialization:

یک جمعیت متشکل از ۱۰۰ ذره ایجاد میکنیم. برای هر ذره، یک بردار مختصات (x1, x2) تصادفی و یک بردار سرعت (v1, v2) تصادفی انتخاب میکنیم. مقادیر C1 و c2 و w را برای آپدیت کردن ذره ها طبق فرمول زیر، انتخاب میکنیم. مقادیر Pbest و Gbest هر ذره را هم برابر منفی بی نهایت قرار میدهیم.

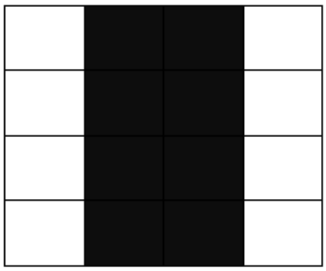
$$v_i = W v_i + c_1 r_1 (P_{best,i} - x_i) + c_2 r_2 (g_{best} - x_i)$$

۲- Fitness:

برای محاسبه فیتنس یک تصویر، تصویر را روی تصویر هدف(برای من تصویر شماره ۱) قرار میدهیم و تعداد پیکسل های مشترک را میشماریم.



برای مثال، فیتنس این عکس برابر ۱۰ است:



۳- update:

پس از محاسبه فیتنس ذره ها، مقادیر Gbest و Pbest ذره ها به روز رسانی میشود. سپس موقعیت جدید ذره ها را به دست میاوریم:

$$v_i = W v_i + c_1 r_1 \left(P_{best,i} - x_i\right) + c_2 r_2 \left(g_{best} - x_i\right)$$
$$x_i = x_i + v_i$$

v_i Velocity of the i^{th} particle $P_{best,i}$ Personal best of the i^{th} particle

x_i Position of the i^{th} particle g_{best} Global best

r_1 and r_1 Random numbers c_1 and c_1 Acceleration Coefficients

W Inertia weight

۴- شرط پایان:

مراحل ۲ و ۳ تکرار میشوند تا وقتی شرط پایان براورده شود. شرط پایان میتواند رسید به فیتنس دلخواه یا پیش رفتن تا حداکثر ۵۰ مرحله باشد.

به کمک این روش، ابتدا ذره ها به صورت پراکنده پخش هستند اما به مرور زمان به علت بیشتر بودن فیتنس تصویر هدف، به مرور تعداد ذره ها به سمت تصویر هدف بیشتر میشود.