به نام خالق رنگین کمان

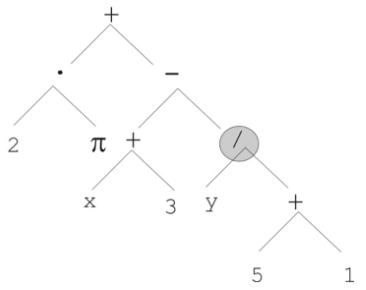
ستاره باباجانی – 99521109

سوال 1:

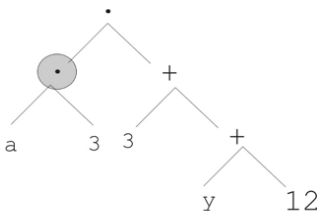
قسمت 1) در ابزاری که از الگوریتم‌های برنامه‌نویسی ژنتیک برای طراحی صفحات وب استفاده می‌کند، یک سری توضیحات از کاربر درمورد طراحی وب مورد نظر جمع آوری میکند(مثل functionality requirements, design constraints, layout preference, ...) و درنهایت راه‌حل‌های کاندید به صورت درخت نشان داده می‌شوند که هر درخت مربوط به یک طراحی صفحه وب خاص است.

- مقداردهی اولیه: جمعیت اولیه از درختان تولید شده به طور تصادفی که طرح‌های صفحه وب را نشان می‌دهند، ایجاد میکنیم.(هر گره از درخت میتواند یک تگ html باشد و شاخه ها نشان دهنده تو در تو بودن این تگ ها باشند.)
- ارزیابیfitness : تناسب هر درخت را بر اساس معیارهای تعریف شده توسط کاربر(مثل جذابیت بصری، پایبندی به ترجیحات طراحی، پاسخگویی) ارزیابی میکنیم.
- انتخاب selection: درختان بعنوان والد برای نسل بعد بر اساس fitness score شان انتخاب میشوند.

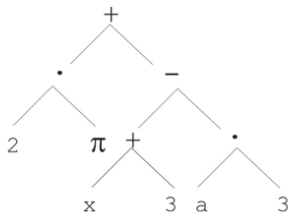
- Cross over: عناصر دو درخت والد را برای ایجاد فرزندان ترکیب میکنیم. مانند مثال گفته شده در جزوه:



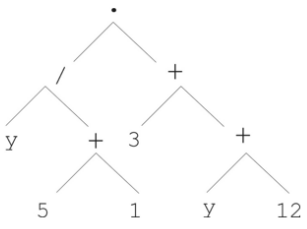
Parent 1



Parent 2



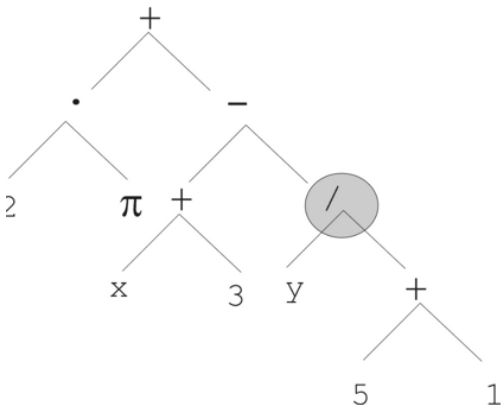
Child 1

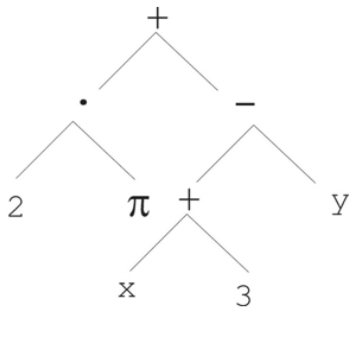


Child 2

18

- mutation: برای کشف احتمالات طراحی جدید، تغییرات تصادفی را در یک درخت معرفی میکنیم. مانند مثال گفته شده در اسلاید:





19

- تکرار: مراحل را از ابتدا دوباره تکرار میکنیم که باعث ایجاد نسل های جدید و درخت ها و طرح های بهتر میشود.

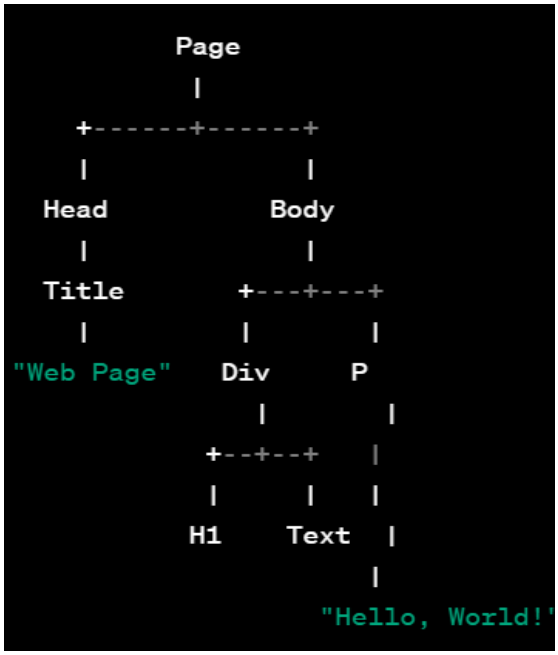
- اتمام: این فرآیند را برای تعدادی مشخص نسل یا تا زمانی که معیارهای همگرایی برآورده شوند، تکرار میکنیم.(شرط توقف از قبل باید مشخص شود.)

به طور کلی همان طور که در اسلاید ها نیز گفته شده بود، در برنامه‌ریزی ژنتیکی، درخت‌ها اغلب به صورت AST یا درخت‌های بیان نشان داده می‌شوند که هر گره در درخت مربوط به یک تابع یا عملیات است و برگها عملوندها یا مقادیر پایانی را نشان می دهند اما در زمینه طراحی صفحه وب، گره ها می توانند عناصر HTML، ویژگی های CSS یا توابع جاوا اسکریپت را نشان باشند.

یک نمونه مثال از ساختار درخت در شکل زیر شرح داده شده است(یک صفحه وب با یک عنوان ("Web Page") و یک پاراگراف ("Hello, World!")) که در آن ریشه نمایانگر صفحه وب به عنوان کلی است و دو گره Head و Body وجود دارند که به ترتیب نمایانگر بخش Head صفحه است که در آن یک گره Title وجود دارد و نمایانگر بخش Body صفحه که شامل یک گره Div و یک گره P است، هستند.

- گره Title نمایانگر عنوان صفحه ("Web Page") است.
- گره Div نمایانگر یک عنصر در صفحه است که خودش شامل دو گروه H1 و Text میشود.
- گره H1 نمایانگر یک عنصر <h1> است.

- گره Text نمایانگر متن ثابت است.
- گره P نمایانگر یک عنصر <p> در صفحه است.



قسمت 2) تابع fitness ارزیابی می کند که صفحه وب تولید شده چقدر نیازهای کاربر را برآورده می کند که در آن باید جنبه هایی مانند جذابیت بصری، پایبندی به ترجیحات طراحی، پاسخگویی در نظر گرفته شود.

در طراحی تابعfitness باید رویکرد های زیر درنظر گرفته شوند:

- حضور و قرارگیری عنصر: پاداش برای وجود عناصر مورد نیاز (فرم، کادرهای ورودی، دکمه) و جریمه برای عناصر نابجا یا غیر ضروری.
- طراحی ظاهر: پاداش برای یک ظاهر طراحی جذاب و منسجم (رنگ، فونت، فاصله) و جریمه برای سبک های ناسازگار یا بیش از حد پیچیده.(مثلا استفاده از ویژگی های css)

- عملکرد: پاداش برای عناصر عملکردی خواسته شده و جریمه برای ویژگی های غیر کاربردی یا نادرست.

شمای کلی از رویکرد های مورد نظر در کد زیر آمده که حاصل جمع پاداش های سه قسمت گفته شده و تفریق آنها از مقدار جریمه به دست آمده است(هرکدام را در **W** منحصر به فردی ضرب کردیم تا اهمیت نسبی معیار های مختلف بتوانند لحاظ شوند.):

```
def evaluate_fitness(input_web_page):
    #1
    existence_reward = calculate_existence_reward(input_web_page)
    #2
    styling_reward = calculate_styling_reward(input_web_page)
    #3
    functionality_reward = calculate_functionality_reward(input_web_page)
    #4
    violation_score = calculate_violations(input_web_page)

    whole_reward = w1*existence_reward + w2*styling_reward + w3*functionality_reward - w4*violation_score
    return whole_reward
```

دلایل انتخاب:

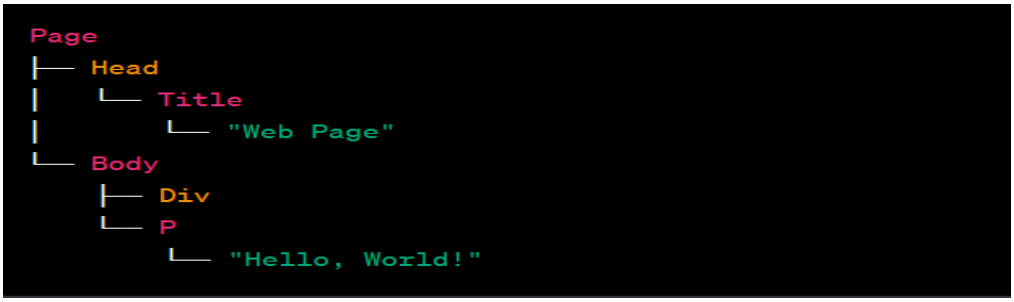
- انطباق با الزامات کاربر: اولین پاداش تضمین می کند که صفحه وب ایجاد شده شامل عناصر مورد نیاز مشخص شده توسط کاربر (به عنوان مثال، فرم ثبت نام، کادرهای ورودی، دکمه) است. این برای برآوردن نیازهای صریح کاربر ضروری است.
- جذابیت بصری و انسجام: پاداش دوم جنبه های بصری صفحه وب را ارزیابی می کند و بر اهمیت طراحی جذاب و منسجم از نظر بصری تأکید می کند.

- مناسب بودن از نظر عملکرد: پاداش سوم جنبه های تعاملی صفحه وب را ارزیابی می کند. این تضمین می کند که صفحه وب نه تنها خوب به نظر می رسد بلکه همانطور که در نظر گرفته شده است عمل می کند.

قسمت 3) ورودی داده شده: 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.

برای رسیدن به جواب خوسته شده از جمعیت اولیه با 3 درخت رندوم آغاز میکنیم. درخت ها به شرح زیر هستند:

- درخت اول: مقدار فیتنس آن: کم(زیرا در 3 پاداش گفته شده مقدار کمی دارد و المان هایی را miss کرده و استایل دهی خاصی ندارد.)



- درخت دوم: مقدار فیتنس آن: معمولی(زیرا پاداش اول زیادی دارد چون اکثر المان های ضروری را دارد، پاداش دوم کمی دارد چون استایل خاصی ندارد و همچنین پاداش سوم کمی دارد.)

- درخت سوم: مقدار فیتنس: زیاد(زیرا پاداش اول زیادی دارد چون همه ی المان ها را پوشش داده است، پاداش دوم کمی دارد چون استایل خاصی داده نشده و پاداش سوم کمی دارد.)

پس طبق population و مقادیر فیتنس، بهترین درخت، درخت سوم است. الگوریتم احتمالا درخت دوم و سوم را برای mutation , cross-over انتخاب کند(چون بهترین ها هستند.) سپس این روند هی تکرار میشود و نسل های جدید و بهتر ساخته میشوند.

جواب نهایی به شرح زیر شد:

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Registration Form</title>

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

        <label for="surname">Surname:</label>
        <input type="text" id="surname" name="surname" required>

        <button type="submit">Complete Registration</button>
    </form>
</body>
</html>
```

سوال 2: کد زده شده شامل دو کلاس اصلی است که در ادامه به بررسی هر کدام میپردازیم:

1) کلاس کروموزوم: در این کلاس ساختار هر کروموزوم شکل میگیرد. هر کروموزوم شامل 6 عدد(6 ژن) میباشد و در تابع __init__ کروموزوم شکل میگیرد. در تابع دوم، یک عدد اعشاری از ژن های کروموزوم بدست می آورد تا بتواند مقدار فیتنس این کروموزوم را حساب کند. در تابع فیتنس طراحی شده، امتیاز بیشتر را به کروموزوم هایی که نزدیک به صفر هستند میدهد.(برعکس مقدار ارزش معادله)

```python
class Chromosome:
    def __init__(self, genes):
        self.genes = genes  # each chromosome has 6 numbers

    # Convert the six-part gene into a float number
    def number_of_each_population(self):
        integer_part = self.genes[0] * 100 + self.genes[1] * 10 + self.genes[2]
        decimal_part = self.genes[3] * 0.1 + self.genes[4] * 0.01 + self.genes[5] * 0.001
        return integer_part + decimal_part

    def fitness(self, equation):
        x = self.number_of_each_population()
        return 1 / (1 + abs(equation(x)))
```

2) کلاس اصلی الگوریتم ژنتیک: در کلاس تعریف شده، مراحل مختلف الگوریتم ژنتیک تعریف میشوند. ابتدا در تابع ___init___ سایز population و معادله ورودی داده میشود و سپس جمعیت اولیه به صورت رندوم از طریق تابع دوم که ژن های تصادفی برای کروموزوم ها ایجاد میشوند، ساخته میشوند.

در تابع _select_parents عملیات selection رخ میدهد و از طریف روش tournament گفته شده در اسلاید های درس، بهترین کروموزوم ها بر اساس مقدار فیتنسشان انتخاب میشوند. دو تابع بعدی برای reproduction هستند که به ترتیب در تابع cross_over، cross_over تک نقطه ای که در اسلاید ها نیز آمده انجام میشود(تبادل ژن بین دو کروموزوم) و در تابع جهش، یک ژن کروموزوم بصورت تصادفی انتخاب شده و تغییر میکند. در تابع بعدی عملیات های گفته شده صدا زده میشوند و سپس بر اساس جواب بدست آمده(تابع

run)، نسل جدید ساخته میشود و دوباره همه ی فرآیند از اول تکرار میشود.(در این تابع خروجی نیز نشان داده میشود.)

```python
class GeneticAlgorithm:
    def __init__(self, population_size, equation):
        self.population_size = population_size
        self.equation = equation
        self.population = self._initialize_population()

    def _initialize_population(self):
        return [Chromosome([random.randint(0, 9) for _ in range(6)]) for _ in range(self.population_size)]

    def _select_parents(self):
        # Tournament selection
        parents = []
        for _ in range(self.population_size // 2):
            random_candidates = random.choices(self.population, k=2)
            parent = max(random_candidates, key=lambda c: c.fitness(self.equation))
            parents.append(parent)
        return parents

    def _crossover(self, parent1, parent2):
        # Single-point crossover
        crossover_point = random.randint(1, 5)
        child1_genes = parent1.genes[:crossover_point] + parent2.genes[crossover_point:]
        child2_genes = parent2.genes[:crossover_point] + parent1.genes[crossover_point:]
        return Chromosome(child1_genes), Chromosome(child2_genes)
```

```python
def _mutate(self, chromosome):
    # Randomly mutate a gene
    mutation_point = random.randint(0, 5)
    chromosome.genes[mutation_point] = random.randint(0, 9)
    return chromosome

def _create_new_generation(self, parents):
    new_generation = []
    for _ in range(0, len(parents), 2):
        parent1, parent2 = parents[_], parents[_ + 1]
        child1, child2 = self._crossover(parent1, parent2)
        new_generation.extend([self._mutate(child1), self._mutate(child2)])
    return new_generation

def run(self, generations=100):
    for _ in range(1000):
        parents = self._select_parents()
        self.population = self._create_new_generation(parents)

        # Sorted by fitness and print best solution
        self.population.sort(key=lambda c: c.fitness(self.equation), reverse=True)
        best_solution = self.population[0]
        print(f"Best solution: {best_solution.genes} -> {best_solution.number_of_each_population()} with fitness {best_solution.fitness(self.equation)}")

        if best_solution.fitness(self.equation) >= 0.95:  # termination
            break

    return self.population[0]
```

حال به بررسی چندجمله ای های داده شده میپردازیم:

```python
def equation(x):
    return 2 * x - 4

# Run the algorithm
population_size = 100
genetic_algorithm = GeneticAlgorithm(population_size, equation)
solution = genetic_algorithm.run()
print("\n")
print(f"Found solution: {solution.number_of_each_population()} with genes {solution.genes}")
print(f"The result of the best chromosome is {equation(solution.number_of_each_population())}")
print("\n")
```

نتیجه:

```
Best solution: [0, 0, 6, 0, 9, 8] -> 6.098 with fitness 0.10874293170943888
Best solution: [0, 0, 6, 4, 9, 5] -> 6.495 with fitness 0.10010010010010009
Best solution: [0, 0, 2, 0, 0, 9] -> 2.009 with fitness 0.982318271119843


Found solution: 2.009 with genes [0, 0, 2, 0, 0, 9]
The result of the best chromosome is 0.017999999999999794
```

```python
# Define your equation here
def equation(x):
    return x * x - 8 * x + 4

# Run the algorithm
population_size = 100
genetic_algorithm = GeneticAlgorithm(population_size, equation)
solution = genetic_algorithm.run()
print("\n")
print(f"Found solution: {solution.number_of_each_population()} with genes {solution.genes}")
print(f"The result of the best chromosome is {equation(solution.number_of_each_population())}")
print("\n")
```

نتیجه:

```
Best solution: [0, 0, 0, 5, 0, 9] -> 0.509 with fitness 0.8424024982288487
Best solution: [0, 0, 0, 5, 0, 2] -> 0.502 with fitness 0.809058870359643
Best solution: [0, 0, 0, 5, 0, 2] -> 0.502 with fitness 0.809058870359643
Best solution: [0, 0, 0, 4, 9, 8] -> 0.498 with fitness 0.7911367369090605
Best solution: [0, 0, 7, 4, 8, 8] -> 7.488 with fitness 0.85752702925196
Best solution: [0, 0, 0, 4, 9, 0] -> 0.49 with fitness 0.7575183698204682
Best solution: [0, 0, 0, 5, 3, 3] -> 0.533 with fitness 0.9803066203046993


Found solution: 0.533 with genes [0, 0, 0, 5, 3, 3]
The result of the best chromosome is 0.02008899999999958
```

```python
# Define your equation here
def equation(x):
    return 4 * (x**3) + 5 * (x**2) + x - 1

# Run the algorithm
population_size = 100
genetic_algorithm = GeneticAlgorithm(population_size, equation)
solution = genetic_algorithm.run()
print("\n")
print(f"Found solution: {solution.number_of_each_population()} with genes {solution.genes}")
print(f"The result of the best chromosome is {equation(solution.number_of_each_population())}")
print("\n")
```

نتیجه:

```
Best solution: [0, 0, 1, 3, 8, 0] -> 1.3800000000000001 with fitness 0.04669779354793397
Best solution: [0, 0, 1, 3, 2, 2] -> 1.322 with fitness 0.05180763846715398
Best solution: [0, 0, 1, 1, 9, 5] -> 1.195 with fitness 0.0659583422280906
Best solution: [0, 0, 1, 5, 9, 1] -> 1.591 with fitness 0.032941898713888994
Best solution: [0, 0, 1, 0, 6, 7] -> 1.067 with fitness 0.08606951141818675
Best solution: [0, 0, 0, 3, 3, 8] -> 0.3380000000000001 with fitness 0.9401342373303144
Best solution: [0, 0, 0, 3, 2, 7] -> 0.32700000000000007 with fitness 0.9984941390371054


Found solution: 0.32700000000000007 with genes [0, 0, 0, 3, 2, 7]
The result of the best chromosome is 0.0015081320000003284
```

```python
# Define your equation here
def equation(x):
    return 183 * (x**3) - 7.22 * (x**2) +15.5 *  x  -13.2

# Run the algorithm
population_size = 100
genetic_algorithm = GeneticAlgorithm(population_size, equation)
solution = genetic_algorithm.run()
print("\n")
```

نتیجه:

```
Best solution: [0, 0, 0, 1, 4, 6] -> 0.14600000000000002 with fitness 0.0867951534577857
Best solution: [0, 0, 0, 4, 3, 7] -> 0.43700000000000006 with fitness 0.11810997044048885
Best solution: [0, 0, 0, 3, 9, 0] -> 0.39 with fitness 0.2776069723767181
Best solution: [0, 0, 0, 3, 9, 6] -> 0.396 with fitness 0.23981215793769345
Best solution: [0, 0, 0, 2, 4, 0] -> 0.24000000000000002 with fitness 0.11953029375764997
Best solution: [0, 0, 0, 3, 6, 0] -> 0.36000000000000004 with fitness 0.9826426010942744


Found solution: 0.36000000000000004 with genes [0, 0, 0, 3, 6, 0]
The result of the best chromosome is -0.01766399999999635
```

سوال 3:

- ساختار ژنوم : به دلیل اینکه جدول ما 6*6 میباشد ما در هر ژنوم یا کروموزوم یک لیست بصورت اعداد 1 تا 36 داریم که نشان دهنده اعداد هر خانه است.
- تابع فیتنس : بر اساس تعداد سطرها و ستون هایی که اعداد زوج و فرد مساوی دارند تعیین میشود.(هرچه تعداد آنها بیشتر کروموزوم برنده تر و score آن بالاتر) دو روش رایج این تابع بصورت زیر است:

1)$fitness$ = 12 − ($number\ of\ rows\ with\ unequal\ \ odd/even\ +\ number\ of\ columns\ with\ unequal\ odd/even$)

2)$total\ difference$ = $Sum\ of\ absolute\ across\ all\ rows\ and\ columns\ fitness$ = 1 / (1 + $total\ difference$)

- پارامترها : از اندازه جمعیت 100(یا 200) تایی شروع میکنیم زیرا مسئله نسبتا ساده است و از روش selection، roulette wheel استفاده میکنیم. برای cross over، از روش cross over دو نقطه ای یا Order1 cross over که در جزوه آمده است، استفاده میکنیم.
- نرخ cross over : 0.7 تا 0.9 و نرخ mutation : 0.1 تا 0.01 میگیریم.
- برای mutation هم از روش swap mutation که در جزوه آمده است، استفاده میکنیم به این صورت که دو المان از یک آرایه را با هم تعویض میکنیم.
- تعداد نسل: چون مسئله زیاد پیچیده نیست، 1000 درنظر میگیریم.

مراحل الگوریتم ژنتیک برای مسئله گفته شده، به این صورت است:

1. ایجاد جمعیت اولیه: ایجاد یک جمعیت اولیه از کروموزوم‌ها، هر کروموزوم با استفاده از لیست اعداد ۱ تا ۳۶ بصورت تصادفی مختلف ایجاد می‌شود.
2. محاسبه مقدار فیتنس: تابع فیتنس که معیار سلامتی را بر اساس تعداد سطرها و ستون‌هایی که اعداد زوج و فرد مساوی دارند تعیین می‌کند.

3. انتخاب: با استفاده از روش Roulette Wheel، والدین بر اساس امتیاز فیتنس انتخاب می‌شوند. والدین با امتیاز فیتنس بالاتر احتمال بیشتری برای انتخاب دارند.
4. Reproduction: از روش Crossover برای ترکیب ژن‌های والدین و ایجاد فرزندان استفاده میکنیم. همچنین جهش با نرخ ۰.۱ تا ۰.۰۱ روی کروموزوم‌ها انجام میدهیم.
5. تکرار: مراحل 2 تا 4 تا رسیدن به شرایط پایانی ادامه می‌یابد.
6. اتمام الگوریتم: رسیدن به بهترین راه حل، رسیدن به حد تعداد مشخص از نسلها.

سوال 4: در این سوال با توجه به شماره دانشجویی بنده حالت اول هستم.

- در ACO، مورچه‌ها به صورت هماهنگ با یکدیگر در جستجوی بهینه‌ترین مسیر حرکت می‌کنند. در اینجا، ما می‌توانیم از ACO برای تشخیص تصاویر حالت اول استفاده کنیم. یعنی موقعیت‌های مورچه‌ها می‌توانند نشان‌دهنده مقدارهای مختلف پیکسل‌ها در تصویر باشند. هر مورچه به عنوان یک حلقه حرکت می‌کند و بر اساس مقادیر پیکسل‌ها در مسیر خود، احتمال تعلق یک تصویر به حالت اول را محاسبه می‌کند.
- برای استفاده از الگوریتم ACO در تشخیص تصاویر با حالت اول، یک گراف تعریف میکنیم. در اینجا، هر گره از گراف نمایانگر یک تصویر و هر

یال نمایانگر ارتباط بین دو تصویر است. همچنین، هر گره ممکن است دارای یک مقدار معنایی (احتمال تعلق به حالت اول) باشد.

سوال گفته تعداد تصویر 5 تا است، گراف گفته شده به شرح زیر میشود:

1. هر گره از گراف نمایانگر یک تصویر میباشد.
2. هر یال بین دو گره نشان‌دهنده ارتباط بین دو تصویر میباشد.(به یال ها مقادیر فرمون ها داده میشود.)
3. هر گره ممکن است دارای یک ویژگی معنایی باشد که نشان‌دهنده احتمال تعلق به حالت اول است. این ویژگی می‌تواند مقداری عددی باشد.

در هر مرحله از الگوریتم ACO، مقدار معنایی هر گره با توجه به مقادیر مجاورانش تغییر می‌کند(میزان فرومون ها)، و مورچه‌ها بر اساس این مقادیر احتمال تصمیم‌گیری می‌کنند.

- هر تصویر را به عنوان یک گره در گراف الگوریتم ACOدر نظر میگیریم و گره ها به وسیله یال ها و میزان فرومون های مختلف به هم متصل میشوند و آنها را بر اساس شباهتاشان به حالت اول مقایسه میکنیم.(از طریق تابع heuristic)
- همان طور که در اسلاید های درس نیز آمده، تعداد فرومون بر روی هر یال افزایش یا کاهش احتمال انتخاب مسیر توسط مورچه‌ها را تعیین

می‌کند. مورچه‌ها تمایل دارند به سمت مسیرهایی حرکت کنند که فرومون‌های بیشتری دارند.

توضیحات نحوه انجام این الگوریتم برای این سوال به شرح زیر است:

1. مقدار دهی اولیه مورچه ها: هر مورچه از یک گره (یا مکان) خاص در فضای جستجو شروع به حرکت می‌کند.(ممکن است همه مورچه‌ها در ابتدا از یک گره شروع کنند یا از گره‌های مختلف)
2. انتخاب مسیر: مورچه با توجه به اطلاعات موجود (فرومون‌ها و مقدار heuristic بدست آمده) تصمیم می‌گیرد که به کدام گره حرکت کند. این مقدار heuristic میتواند شباهت بین تصویر فعلی و حالت اول باشد که این similarity به روش های مختلفی بدست بیاید.(میتوان فاصله را بصورت معکوس مقدار similarity درنظر گرفت.)
همان طور که در اسلاید درس آمده، احتمال انتخاب یک گره j توسط یک مورچه k معمولا به صورت زیر محاسبه میشود:

$$P_{k,j} = \frac{\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}}{\sum_{l \in allowedMoves} \tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}}$$

3. حرکت به گره بعدی: پس از انتخاب مسیر، مورچه به گره بعدی حرکت می‌کند و اطلاعات فرومون‌ها در این گره به‌روزرسانی می‌شود.(فرمون بیشتری در یال هایی که مورچه ها از انها بازدید میکنند جمع میشوند).

همان طور که در اسلاید درس آمده، تابع بروز رسانی فرمون به شرح زیر است:

$$\tau_{ij} = (1-\rho)\cdot\tau_{ij} + \Delta\tau_{ij}$$

$$\Delta\tau_{ij} = \sum_{k=1}^{n} \frac{1}{L_k}$$

4. مراحل قبل را به تعداد مشخصی تکرار میکنیم تا به همگرایی برسیم و میانگین امتیاز شباهت مسیرهای مورچه را در نظر میگیریم و تصاویری که بیشتر شبیه حالت اول (با استفاده از امتیازشان) باشند به عنوان تصویر اول شناخته میشوند.
5. پایان حرکت: حرکت مورچه تا زمانی ادامه پیدا می‌کند که یک شرایط خاتمه معین شود (به عنوان مثال، تعداد مراحل مشخص یا رسیدن به یک حالت مطلوب).

پایان