توضیحات سوال دوم پروژه اول درس هوش مصنوعی

با توجه به اینکه این سوال با جاوا پیاده سازی شده است، توضیح مختصری از کلاس ها(با رنگ زرد) و برخی فیلدها(با رنگ سبز) و متدهای آن(با رنگ بنفش) آورده شده است:

- کلاس Graph: در اینجا، گراف خود را به کمک مجموعه گره ها(vertices) و یال ها(edges) تعریف می کنیم. همچنین، رنگ ها نیز به صورت private static final int تعریف شده است تا بتوان به راحتی با آن ها کار کرد.(رنگ ها را می توان بسته به نیاز، کم یا زیاد و یا عوض کرد)
در نهایت نیز، برای این گره ها، درجه(degree) و رنگ(color) شان را نیز نگه می داریم.

```
public class Graph{
    private static final int RED = 1;//color red
    public static final int YELLOW = 2;//color yellow
    private static final int GREEN = 3;//color green
    private static final int BLUE = 4;//color blue
    private ArrayList<String> vertices;//set of vertices
    private ArrayList<String[]> edges;//from one vertice to another vertice
    private ArrayList<Integer> degrees;//degree correspondent to that node
    private ArrayList<Integer> colors;//colors correspondent to that node
```

پ.ن: در کلاس Main، مجموعه استان ها و یال های ایران آورده شده است که در ادامه شرح داده شده است.

- کلاس Genetic: در اینجا، یک گراف(graph) می گیریم و روی آن، الگوریتم ژنتیک را اعمال می کنیم. سایر فیلدها، شامل اندازه جمعیت اولیه(population_size)، اندازه مسابقه (tournament_size) ، نرخ جهش(mutation rate) و تعداد نسل (number_of_generations) می باشد.

```
public class Genetic{
    private static final int RED = 1;//color red
    private static final int YELLOW = 2;//color yellow
    private static final int GREEN = 3;//color green
    private static final int BLUE = 4;//color blue
    private int population_size;//number of chromosomes
    private int tournament_size;//number of chromosomes in a tournament
    private double mutation_rate;//the rate of mutation
    private int number_of_generations;//number of times we simulate the genetic algorithm!
```

اجرای الگوریتم نیز بدین شکل است که پس از ایجاد جمعیت اولیه(create_first_population)، یک for به اندازه تعداد نسل داریم که در هر نسل، می آید و والدین را از طریق تابع گزینش از طریق مسابقه(tournament_selection) و با در نظر داشتن تابع برازش(fitness_function)، انتخاب می کند.

سپس تابع تولید نسل جدید(generate_new_population) فراخوانی می شود که وظیفه crossover کردن والدین و تولید فرزند را برعهده دارد.

نهایتا نیز، بر روی نسل جدید، تابع اعمال جهش(apply_mutation) فراخوانی می شود که مطابق با نرخ جهش، روی کروموزوم ها جهش اعمال می کند.

```
create_first_population();

for (int i = 0; i < number_of_generations; i++) {
    parent_chromosomes.clear();//clearing the previous round
    System.out.println("generation #" + (i + 1));
    tournament_selection();

    new_generation = generate_new_population();
    new_generation = apply_mutation();

    chromosomes = new_generation;//getting ready for the next round!
}
System.out.println("genetic algorithm simulated successfully!");
```

پس از اتمام الگوریتم، بهترین، بدترین و میانگین تابع برازش به ازای هر نسل چاپ شده و یک نمونه رنگ آمیزی مطابق با پاسخ مسئله نمایش داده می شود.

```
wait(2000);
print_best_worst_and_average_fitness();//prints the best, worst and average fitness of every generation
wait(3000);
for (Graph chrom : best_fitness) {
    if(this.fitness_function(chrom) == 1.0){
        System.out.println("an example of a solved graph is: ");
        chrom.show_colors();
        break;
    }
}
```

- کلاس Annealing: در اینجا، دوباره یک گراف(graph) می گیریم و با در نظر گرفتن همسایه ها(neighbours) روی آن، الگوریتم ذوب فلزات را اعمال می کنیم. سایر فیلدها شامل دمای

اولیه(T0)، ضریب کاهش دما(α)، مدل کاهش دما(mode)، و تعداد اجرای الگوریتم(round با مقدار اولیه 1) می باشد.

همچنین، مدل کاهش دما نیز برای راحتی کار به صورت private static final int تعریف شده است.(مثلا EXPONENTIAL COOLING برابر 1 می باشد)

```
public class Annealing{
    private Graph graph;
    private ArrayList<Graph> neighbours;
    private double t0;//initial temperature
    private double t;// current temperature
    private double alpha;//The coeffecient used for temperature cooling
    private static final int EXPONENTIAL_COOLING = 1;
    private static final int LOGARITHMIC_COOLING = 2;
    private static final int NEGATIVE_ONE_ORDER_COOLING = 3;
    private static final int NEGATIVE_TWO_ORDER_COOLING = 4;
    private int mode;
    private int round = 1;//number of the round
```

اجرای الگوریتم نیز بدین شکل است که مادامی که به حالت هدف نرسیدیم یا دما تقریبا برابر صفر نشده است، همسایه های آن حالت را از طریق تابع یافتن همسایه ها (find_neighbours) پیدا کرده و سپس یکی را به تصادف انتخاب می کنیم. اگر بهتر بود، به آن می رویم و اگر بدتر بود، با احتمال $e^{\frac{\Delta E}{T}}$(که ΔE اختلاف fitness_function دو حالت می شود) به حالت همسایه می رود.

در انتها نیز، یک واحد به تعداد اجرای الگوریتم اضافه می شود و کاهش دما از طریق فراخوانی تابع change_temperature صورت می گیرد.

```
while(this.fitness_function(graph) != 1){//we haven't reached the goal
    System.out.println("initial graph fit: " + this.fitness_function(graph));
    System.out.println("round #" + (round));
    neighbours = find_neighbours();

    for (int i = 0; i < neighbours.size(); i++) {
        System.out.println(this.fitness_function(neighbours.get(i)));
    }

    go_to_neighbour();

    round++;
    t = change_temperature(alpha, round, mode);
}
System.out.println("annealing finished in " + round + " rounds");
```

❖ کلاس Main: در اینجا، ابتدا استان ها و ارتباط های بین استان ها در یک آرایه ریخته شده است.
استان ها به صورت زیر نامگذاری شده اند:

Symbol	Province	استان
W	West Azerbaijan	آذربایجان غربی
E	East Azerbaijan	آذربایجان شرقی
A	Ardabil	اردبیل
K	Kurdistan	کردستان
Z	Zanjan	زنجان
G	Gilan	گیلان
K’	Kermanshah	کرمانشاه
H	Hamadan	همدان
Q	Qazvin	قزوین
I	Ilam	ایلام
L	Lorestan	لرستان
M’	Markazi	مرکزی
A’	Alborz	البرز
M	Mazandaran	مازندران
G’	Golestan	گلستان

N	North Khorasan	خراسان شمالی
Q’	Qom	قم
T	Tehran	تهران
S	Semnan	سمنان
R	Razavi Khorasan	خراسان رضوی
U	Khuzestan	خوزستان
C	Chaharmahal and Bakhtiari	چهارمحال و بختیاری
I’	Isfahan	اصفهان
S’	South Khorasan	خراسان جنوبی
B’	Kohgiluyeh and Boyer-Ahmad	کهگیلویه و بویر احمد
Y	Yazd	یزد
B	Bushehr	بوشهر
F	Fars	فارس
J	Kerman	کرمان
H’	Hormozgan	هرمزگان
T’	Sistan and Baluchestan	سیستان و بلوچستان

سپس، از کاربر می خواهیم که از بین الگوریتم های ژنتیک یا ذوب فلزات، یکی را انتخاب کرده و با دادن پارامترهای مورد نیاز، الگوریتم خواسته شده اجرا می شود و نتایج آن چاپ می شود.

نتایج:

- ✓ **الگوریتم ژنتیک:** نتایج تست کردن پارامترهای الگوریتم ژنتیک با مقدارهای مختلف در فایل genetic_results.txt آورده شده است. درباره همگرایی هر پارامتر داریم:
  - **تعداد نسل:** هر چه تعداد نسل بیشتر باشد، الگوریتم بیشتر اجرا می شود و بنابراین، احتمال رسیدن به پاسخ مسئله در انتهای اجرا، بیشتر می شود.

- **جمعیت اولیه:** هر چه جمعیت اولیه بیشتر باشد، احتمال وجود کروموزوم های بهتر بیشتر می باشد و در نتیجه، زودتر می توان به پاسخ مسئله رسید.
- **سایز مسابقه:** اگر سایز مسابقه زیاد باشد، کروموزوم شایسته تری انتخاب می شود اما تعداد والدین کمتر می باشد(طبق فرض مسئله) از طرفی، اگر سایز مسابقه کم باشد، کروموزوم های بیشتری به عنوان والد داریم اما مثل کروموزوم های حالت قبل، شایسته نیستند. بنابراین، سایز مسابقه در یک نقطه میانی، باعث بهینه کردن الگوریتم می شود.
- **نرخ جهش:** هر چه نرخ جهش بیشتر باشد، احتمال اینکه یک کروموزوم تغییر کند بیشتر است. حال این تغییر می تواند مثبت باشد یا منفی. ولی در کل، به سریعتر رسیدن به پاسخ مسئله کمک می کند.

✓ **الگوریتم ذوب فلزات:** نتایج تست کردن پارامترهای الگوریتم ژنتیک با مقدارهای مختلف در فایل annealing_results.txt آورده شده است. درباره تاثیر هر پارامتر داریم:

- **دمای اولیه:** هر چه دمای اولیه بالاتر باشد، احتمال انجام دادن حرکات بد در گام های اولیه بیشتر می باشد
- **ضریب کاهش دما:** بستگی به مدل کاهش دما دارد؛ اما در کل هر چه کاهش دما آرام تر انجام پذیرد، احتمال رسیدن به جواب بیشتر می باشد.
- **مدل کاهش دما:** با مشاهده نمودارهای مربوط به کاهش دما، می بینیم که هر چه دما آرام تر کاهش پیدا کند(مانند مدل دوم کاهش دما)، احتمال رسیدن به پاسخ مسئله بیشتر می باشد.