تكليف برنامه نويسي SA

مقدمه:

كدي كه در اختيار شما قرار گرفته است شامل كلاس هاي SA_Algorithm، SA_Coordinator_Part_I و Main مي باشد.

كلاس SA_Algorithm

اين كلاس شامل يك تابع start مي باشد كه چارچوب اصلي الگوريتم Simulated Annealing در آن نوشته شده است. در اين تابع از 8 تابع ديگر استفاده شده است كه شما موظفيد اين 8 تابع را با توجه به نوع مسئله خودتان (TSP، OBST و يا Job Scheduling) كامل كنيد.

نكته 1:

از آنجا كه در اين كد براي سادگي كار configuration ها به صورت آرايه از int در نظر گرفته شده اند، بايد توجه داشته باشيد كه لازم است ليست داده هاي ورودي شما به صورت آرايه اي global در كلاس SA_Algorithm موجود باشد، تا شما بتوانيد در توابعي مانند objectiveFunction از آن استفاده كنيد و مقادير مورد نظرتان از آن بدست آوريد.

تابع هايي كه بايد آنها را تكميل كنيد:

- **تابع 1:**

```
createRandomInitialConfig()
```

در اين تابع شما بايد با استفاده از توابعي كه عدد تصادفي ايجاد مي كنند مانند Math.random() در جاوا يك configuration تصادفي با توجه به نوع مسئله خودتان ايجاد كنيد و سپس اين configuration را به صورت يك آرايه از int بازگردانيد (return كنيد). توجه داشته باشيد كه اين آرايه با توجه به مسئله شما يا به صورت bit string (مانند {0,0,1,0,1}) و يا به صورت جايگشتي (مانند {2,4,3,5,1}) مي باشد و طول آن نيز برابر با n شما در مسئله، يعني تعداد داده هاي ورودي خواهد بود.

- **تابع 2:**

```
objectiveFunction(int[] config)
```

در اين تابع شما يك configuration از مسئله را دريافت كرده (به صورت آرايه اي از int) و حاصل مسئله را با توجه به داده هاي ورودي (به نكته ي 1 توجه شود) محاسبه كرده و return مي كنيد.

- **تابع 3:**

```
boltzmannFunction(double e, double e1, double k, int t)
```

در اين تابع شما مقدار تابع بولتزمن را با توجه به فرمول آن و داده هاي ورودي تابع محاسبه مي كنيد.

ورودي هاي تابع عبارتند از:

e مقدار objective value كه تا به حال به عنوان جواب انتخاب شده است.

e1 مقدار objective value كه از آخرين configuration بدست آمده است.

k ثابت دما، (كه به صورت پيش فرض در كد داده شده است).

t مقدار دما (دماي فرضي در الگوريتم) در لحظه ي كنوني

- **تابع 4:**

```
counterDefiner(int T)
```

اين تابع تعيين مي كند كه در هر دما چند بار configuration تغيير داده شده و تست شود. تنها ورودي اين تابع دماي كنوني مي باشد.

- **تابع 5:**

```
changeConfig(int[] config)
```

اين تابع (با توجه به نوع مسئله) با كمي تغيير configurationي را كه به آن داده شده است به يك configuration جديد تبديل مي كند.

- **تابع 6:**

```
isValid(int[] config)
```

اين تابع (با توجه به نوع مسئله) تعيين مي كند كه به طور كلي آيا configurationي كه به آن داده شده است، مي تواند جوابي از مسئله باشد (با توجه به ساير پيش فرض ها و تعاريف مسئله) يا خير و در صورتي كه configuration داده شده به آن قابل قبول باشد مقدار true را return مي كند و در غير اين صورت false.

- **تابع 7:**

```
terminate(int T, int unChangedCounter)
```

اين تابع با توجه به مقادير دو ورودي آن تعيين مي كند كه آيا بايد كار الگوريتم پايان داده شود و يا همچنان ادامه پيدا كند. چنانچه اين تابع true بر گرداند يعني كار الگوريتم بايد تمام شود و اگر false بر گرداند يعني الگوريتم همچنان بايد به كار خودش ادامه دهد.

ورودي هاي اين تابع عبارتند از:

T نشان دهنده ي دماي كنوني مي باشد.

unChangedCounter تعيين مي كند كه **در چند بار تغيير configuration اخير**، configuration جديدي انتخاب نشده است.

- **تابع 8:**

```
reduce(int T)
```

اين تابع تعيين مي كند كه مقدار دما در هر حلقه ي مسئله به چه ميزان كاهش پيدا كند. تنها ورودي اين مسئله دماي كنونيست و خروجي آن نيز مقدار دما پس از كاهش به ميزان لازم مي باشد.

كلاس SA_Coordinator_Part_I

این کلاس یک Coordinator است که با آن می توان نمودار اوّل را رسم کرد.

تابع زیر را از این کلاس کامل کنید:

- **تابع 1:**

```
generateRandomData(int configSize)
```

در این تابع شما می بایست به اندازه و سایز ورودی (configSize) داده های تصادفی برای مسئله خود تولید کنید و آن را در متغیّر های استاتیکی که در کلاس SA_Algorithm تولید کردید، ذخیره نمایید .داده هایی همانند :داده و احتمال دسترسی هر گره در OBST؛ سود، زمان اجرا و موعد هر شغل در JS ؛ ماتریس مجاورت در. TSP