تکلیف برنامه نویسی الگوریتم ژنتیکی

مقدّمه

کُد هایی که در اختیار دارید مشتمل بر ۷ کلاس می باشد (که استفاده از eclipse و به زبان Java نوشته شده اند):

۱. Tuple
۲. Population
۳. GA_Algorithm
۴. Main
۵. GA_Coordinator_Part_I
۶. GA_Coordinator_Part_II
۷. GA_Coordinator_Part_III

شما می بایست به ترتیبی که در این مستند ذکر شده کلاس های فوق را (بسته به نوع آن) بنویسید، یا تکمیل کنید و یا تغییر دهید و یا تنها اجرا کنید. همچنین شما نیاز دارید تا بسته به مسئله ی خود، کلاس های لازم را (که احتمالا" باید به خاطر تکالیف قبلی تاحالا نگاشته باشید) اضافه کنید.

***** نکته:*** توجّه داشته باشید که تکلیف برنامه نویسی پیش رو از تکلیف برنامه نویسی انتگرال – که حالت مشابهی دارد – **بدون شک پیچیده تر** است. زیرا که این کُد برای هر سه نوع مسئله ای که به شما نظیر شده است، پیشبینی شده است. از این رو، شما نه تنها نیاز خواهید داشت که کلاس هایی را احتمالا" به این مجموعه کلاس ها اضافه کنید، بلکه شاید لازم ببینید که نیاز دارید به برخی از کلاس های از پیش نوشته شده، متغیّر ها و یا توابعی را اضافه نمایید.

***** نکته:*** با توجّه به این موضوع که مسائل متفاوتی قرار است روی این قالب کُد پیاده سازی شود، طرّاحی تست (همچون تکلیف انتگرال) که با آن بتوانید جای جای کُد خود را بیازمایید پیچیده بود و در این زمان کم برای تیم آموزشی ناممکن.

تابع getData از کلاس GA_Algorithm

قبل از هر چیز این تابع را پُر کنید. این تابع، قسمتی از الگوریتم نیست، بلکه تنها توسّط آن داده های مختلف از کاربر گرفته می شود و در متغیّرهای لازمه ذخیره می شوند. داده های مسئله را از این طریق از کاربر بگیرید. قسمتی از کُد برای شما نوشته شده است. سعی کنید که آن را تکمیل کنید. همچنین در متغیّر list، بسته به جایگشتی بودن و یا زیرمجموعه ای بودن مسئله تان، به ترتیب، اعداد یک تا n ، و n تا صفر وارد کنید (n تعداد گره ها، شغل ها، و یا شهر هاست).

***** نکته:*** برای پُر کردن این تابع شما قطعا" نیاز خواهید داشت که در صورت لزوم متغیّرها، توابع و احتمالا" کلاس های جدیدی برای شبیه سازی مسئله خود به این مجموعه کُد ها اضافه نمایید.

***** نکته:*** **اکیدا"** پیشنهاد می شود که متغیّرهای لازم را به صورت **static** به همین کلاس اضافه کنید تا بتوانید بعدا" در کلاس های دیگر به آن به راحتی رجوع کنید. متغیّر هایی همانند: داده و احتمال دسترسی هر گره در OBST؛ سود، زمان اجرا و موعد هر شغل در JS؛ ماتریس مجاورت در TSP.

***** نکته:*** پس از پُر کردن این تابع به سراغ سایر توابع در این کلاس نروید! به ترتیب همین مستند پیش روید.

کلاس Tuple

این کلاس، یک کلاس برای ذخیره سازی ساختار داده یک tuple است. هر شیء از این کلاس به دو صورت ممکن است باشد:

۱. اگر مسئله ی شما مسئله جایگشتی باشد، نمایانگر یک جایگشت مانند (1,4,5,2,3) خواهد بود
۲. اگر مسئله ی شما زیرمجموعه ای باشد، نمایانگر یک زیرمجموعه مانند (0,0,1,0,1) خواهد بود.

سه متغیّر در این کلاس وجود دارند:

۱. متغیر configuration که یک آرایه است بسته به اینکه مسئله مذکور جایگشتی و یا زیرمجموعه ای است، به ترتیب، جایگشتی از اعداد یک تا n و یا یک آرایه ای از صفر و یک ها خواهد بود.
۲. متغیر cost که هزینه – و یا معادلا" سود- یک tuple را مشخص می کند. مثلا" اینکه یک جایگشت از گره ها معادل با چه average access time در OBST است و یا اینکه یک زیرمجموعه از شغل ها معادل با چه سودی است.
۳. متغیّر random که می تواند در تولید داده های تصادفی به شما کمک کند.

توابع زیر را از این کلاس کامل کنید:

تابع *getObjectiveFunction*

این متُد مقدار تابع هدف متناظر با Tuple ورودی را محاسبه می کند. یعنی:

۱. برای مسئله OBST: محاسبه Average Access Time بر حسب ورود گره ها به درخت به ترتیب موجود در Tuple
۲. برای مسئله TSP: محاسبه طول دور همیلتونی بر حسب طی کردن شهر ها به ترتیب موجود در Tuple
۳. برای مسئله JS: محاسبه سود حاصل از آن دسته از Job هایی که در Tuple ذکر شده اند. (البته به صورت قرینه؛ به نکته زیر توجّه کنید)

پس از محاسبه تابع هدف، قبل ازاینکه آن رابه صورت خروجی برگردانید، مقدار آن را در متغیر cost مربوط به Tuple ورودی تنظیم کنید.

***** نکته:*** برای محاسبه تابع هدف متناظر با هر Tuple شما نیاز خواهید داشت تا از داده های مسئله – که به صورت static در کلاس GA_Algotithm ذخیره کرده اید – استفاده نمایید.

***** نکته:*** دقّت داشته باشید که قالب کلّی این کُد **مینیمم سازی** است. از این رو کاملا" سازگار با OBST و TSP است. امّا برای مسئله JS که یک مسئله **ماکسیمم سازی** است، شما می بایست **منفی** سود را در نظر بگیرید. به عبارت دیگر، برای مسئله JS سود به دست آمده را در **منفی یک** ضرب کنید تا گویای "هزینه" باشد و نه "سود".

***** نکته:*** برای پُر کردن این تابع شما قطعا" نیاز خواهید داشت که در صورت لزوم متغیّرها، توابع و احتمالا" کلاس های جدیدی برای شبیه سازی مسئله خود به این مجموعه کُد ها اضافه نمایید.

***** نکته:*** توجّه داشته باشید که ممکن است Tuple مذکور **معتبر نباشد**. مثلا" در مسئله JS باید job ها feasible باشند؛ یا در مسئله TSP مسیر یاد شده حقیقتا" وجود داشته باشد (هر چند به علّت کامل بودن گراف، این موضوع در مسئله TSP مطرح نیست). اگر یک Tuple معتبر نبود، مقدار هزینه **مثبت بی نهایت** (یا در زبان جاوا Integer.MAX_VALUE) را به آن نظیر کنید.

تابع *randomTuple*

این تابع می بایست یک Tuple تصادفی بر حسب آرایه ورودی تولید کند. به صورت دقیق تر، شما می بایست اندازه و طول آرایه ورودی (list) را یافته (مثلا" n) و بر اساس یک Tuple تصادفی به طول n تولید کنید. این Tuple بسته به جایگشتی و یا زیرمجموعه ای بودن مسئله شما می بایست جایگشتی از اعداد یک تا n و یا n تا صفر-یک باشد. این تغییرات را در

configuration مربوط به Tuple اعمال کنید. در انتهای این تابع **حتما"** تابع getObjectiveFunction را که قبلا" نوشته بودید، را برای این Tuple صدا کنید تا هزینه ی آن محاسبه و ذخیره شود.

کلاس Population

این کلاس برای ذخیره سازی ساختار داده "جمعیّت" و یا Population است. به عبارت دیگر، هر Population مجموعه ای از Tuple هاست که اصطلاحا" به آن ها کروموزوم می گویند.

تابع زیر را از این کلاس کامل کنید:

تابع *randomPopulation*

در این تابع شما می بایست یک Population تصادفی تولید کنید و به عنوان خروجی قرار دهید. شما باید به تعداد POPULATION_SIZE – که به عنوان یک متغیر استاتیک در کلاس GA_Algorithm آمده است – با استفاده از تابع randomTuple ، Tuple های تصادفی تولید کنید و آنها را به مجموعه کروموزوم های Population اضافه کنید. دقّت کنید که باید از ورودی تابع (list) به عنوان ورودی تابع randomTuple استفاده نمایید.

کلاس GA_Algorithm

این کلاس هسته اصلی الگوریتم ژنتیک است و قسمت های مختلف این الگوریتم در این کلاس تنظیم شده است. دو تابع genetic_function و selection نقش اساسی ایفا می کنند. در تابع genetic_function الگوریتم ژنتیک انجام می شود: در یک حلقه جمعیّت ها و نسل های گوناگون تولید می شوند و در نهایت نسل برتر باقی می ماند و از آن بهترین Tuple چاپ می شود. در تابع selection نسل بعد از روی نسل فعلی تولید می شود.

توابع زیر را از این کلاس کامل کنید:

تابع *terminated*

این تابع تعیین کننده اتمام و یا عدم اتمام حلقه اصلی الگوریتم است و خروجی آن به صورت Boolean است. این تابع باید زمانی true برگرداند که یا به تعداد ی مشخصی (مثلا" ۵۰ بار – شما هم از همین عدد استفاده کنید) در حلقه اصلی الگوریتم ژنتیک پیشرفتی حاصل نشده است (که این مفهوم در متغیر repeat ذخیره می شود)، **و یا** اینکه تعداد نسل های ما از ابتدا تا کنون (متغیّر counter) به تعداد لازم (متغیر POPULATION_NUMBER) رسیده باشد. درغیر این صورت، خروجی تابع باید false باشد.

تابع *createRouletteWheels*

در این تابع شما می بایست مفهوم Roulette Wheel را پیاده سازی نمایید. شما باید آرایه RWeels را – که در ابتدا به صورت استاتیک تعریف شده است – پُر نمایید. با توجّه که کروموزوم ها به ترتیب نزولی مرتّب شده اند (از کروموزوم های بد به کروموزوم های خوب)، Roulette Wheel را به صورت زیر باید پُر کنیم:

$$RWeels[i] = \frac{\sum_{j=1}^{i+1} j}{\sum_{j=1}^{n} j} \qquad 0 \le i < n$$

که در آن n همان تعداد جمعیت یک نسل – یا به عبارتی همان POPULATION_SIZE- است.

تابع selectTheTwoBest

در این تابع باید دو تا بهترین کروموزوم ها را از جمعیّت انتخاب کنیم و در یک آرایه قرار داده و به عنوان خروجی برگردانیم. توجّه کنید که از آن جایی که کروموزوم ها از بد به خوب مرتّب شده اند، دو کروموزوم آخر جمعیّت ما باید به عنوان خروجی برگردانده شوند.

تابع selectChromosomes

در این تابع شما لازم است که بر اساس Roulette Wheel ای که ساخته اید، یک کروموزوم از جمعیت انتخاب کنید و به عنوان خروجی برگردانید. به عبارت دقیق تر شما می بایست یک عدد تصادفی بین صفر و یک تولید کرده (مانند r)، سپس کوچکترین i ای را پیدا کنید که r<RWeels[i] . سپس i اُمین کروموزوم از نسل را به خروجی دهید.

تابع crossover

شما می بایست در این تابع مفهوم Cross-over را پیاده سازی کنید. دو Tuple به ورودی داده شده است؛ عمل cross-over روی این دو انجام می شود؛ دو tuple جدید تولید می شود؛ این دو Tuple جدید را در یک آرایه قرار داده و به خروجی دهید. توجّه داشته باشید که عمل crossover میان مسائل جایگشتی و زیرمجموعه ای متفاوت است.

تابع mutation

شما می بایست در این تابع مفهوم Mutation را پیاده سازی کنید. یک Tuple به ورودی داده شده است؛ عمل mutation روی آن انجام می شود؛ یک tuple جدید تولید می شود؛ این Tuple جدید را به خروجی دهید. توجّه داشته باشید که عمل Mutation میان مسائل جایگشتی و زیرمجموعه ای متفاوت است.

کلاس Main

چیزی لازم نیست از این کلاس کامل کنید. بعد از نوشتن توابع فوق، این کلاس را اجرا کنید و ببینید که چه خروجی ای دریافت می کنید و صحّت برنامه خود را چک کنید.

کلاس GA_Coordinator_Part_I

این کلاس یک Coordinator است که با آن می توان ۹ تا از ۱۸ نمودار اوّل را رسم کرد (نمودارهای شماره های زوج). برای این منظور شما لازم است که یکی سه متغیّر Boolean (mutation, crossover, population) را true کنید و دو تای دیگر را false نمایید.

***** نکته:*** توجّه داشته باشید هر زمان که می خواهید هر یک از Coordinator ها را اجرا کنید، می بایست خطی از GA_Algorithm را که در آن تابع getData واقع شده است (خط ۴۵) به صورت comment در آورید.

تابع زیر را از این کلاس کامل کنید:

تابع generateRandomData

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 GA_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.

کلاس GA_Coordinator_Part_II

این کلاس یک Coordinator است که با آن می توان ۹ تا از ۱۸ نمودار اوّل را رسم کرد (نمودارهای شماره های فرد). برای این منظور شما لازم است که یکی سه متغیّر Boolean (mutation, crossover, population) را true کنید و دو تای دیگر را false نمایید.

***** نکته:*** توجّه داشته باشید هر زمان که می خواهید هر یک از Coordinator ها را اجرا کنید، می بایست خطی از GA_Algorithm را که در آن تابع getData واقع شده است (خط ۴۵) به صورت comment در آورید.

لازم نیست چیزی از این کلاس کامل کنید. این کلاس از همان generateRandomData استفاده خواهد کرد.

کلاس GA_Coordinator_Part_III

برای چهار نمودار باقی مانده، شما خودتان می بایست کلاس های coordinator مربوطه را بنویسید. هرچند، coordinator هایی که تا کنون نوشته اید و دیده اید، کاملا" می تواند برای شما الهامبخش باشد.

***** نکته:*** ممکن است راحتتر باشید که از بیش از یک کلاس برای کشیدن چهار نمودار باقی مانده استفاده کنید. در این صورت اگر لازم بود این کار را انجام دهید و کلاس هایی همچون GA_Coordinator_Part_IV و غیره بسازید.

***** نکته:*** برای کشیدن برخی نمودارها، از آنجا که برخی اطّلاعات لازم ممکن است در کلاس اصلی ذخیره نشوند (همچون اطلاعات مربوط به هر نسل) احتمالا" لازم خواهد شد که دستی به بدنه از پیش نوشته ی کُد ببرید و مواردی را اضافه کنید.