مقدمه

شما در نخستین فاز از «شَرَیان اصلی» (Main Stream) درس هستید. در این شریان اصلی، ما با دو مسئله ی بسیار ساده سر و کار داریم: مسئله کوله پشتی دودویی و مسئله درخت جستجوی دودویی بهینه. در هر مرحله، ما الگوریتم هایی را که یاد می گیریم بر روی این دو مسئله اعمال می کنیم.

برای راحتی کار شما دانشجویان، بدنه اصلی کُد از پیش نوشته شده است. شما تنها کافی است توابعی را که خالی است، پُر کنید. در این فاز به الگوریتم های کورکورانه (Blind) و حریصانه (Greedy) می پردازیم.

نکته یک: زبان کُد از پیش نوشته شده JAVA است که در محیط eclipse JUNO نوشته شده است. عزیزانی که مصرّ هستند تا با زبان دیگری کار کنند، خودشان موظّف هستند از ابتدا کُد بزنند. هرچند، می توانند – بلکه توصیه می شود – از ساختار کُدی که برایشان نوشته شده الهام گیرند.

نکته دو: بهتر است که در جریان باشید که کُدی که در اختیار شما است اینگونه حاصل شده است که ابتدا تمامی کُد نوشته شده و اطمینان حاصل شده است که کُد کار می کند. سپس توابع آن پاک شده است. از این رو، از صحّت کُد مطمئن باشید.

نکته سه: هنگام پُر کردن توابع خالی به میزان فضای خالی توجّه کنید؛ زیرا که میزان فضای خالی با مقدار کدی که احتمالا" باید بزنید متناسب است.

ساختار کلّی کُد

قبل از هر چیز بدانید، شما موظّف هستید که به تمامی کُد مسلّط باشد (حتی اگر خودتان آن را ننوشته باشید). شما با دو پروژه ی JAVA رو برو خواهید بود که هر کدام متناظر با هر مسئله است. در هر پروژه، سه package وجود دارد:

1. Algorithms: که الگوریتم ها در آن نوشته شده اند. شامل دو class برای هر یک از الگوریتم های کورکورانه و حریصانه می باشد.
2. Run: که برای اجرا پیشبینی شده است و شامل سه کلاس می باشد:
   a. TestYourMethods: شما بعد از پُر کردن هر تابع، به این کلاس می آیید و آنچه را که نوشته اید تست می کنید.
   b. Knapsack/OBSTTest: شما بعد از اتمام پُر کردن همه توابع، در اینجا می توانید الگوریتم ها را با داده های متفاوت تست کنید.
   c. Coordinator: این کلاس برای تهیه نمودار ها استفاده می شود.
3. Util: که در آن کلاس های مورد نیاز شما برای مسئله ی مربوطه نوشته شده اند.

مسئله کوله پشتی دودویی

صورت مسئله

مسئله ی کوله پشتی دودویی (یا همان Knapsack 0/1) بدین شرح است که تعدادی اشیاء با وزن ها و قیمت های (سود) متفاوت وجود دارد. یک کوله پشتی با ظرفیت و حجم مشخص داریم که می خواهیم تعدادی از این اجناس را برداریم. کدام اشیاء را برداریم تا وزن اشیاء برداشته شده از ظرفیت کوله پشتی بیشتر نشود و در عین حال، سود حاصله بیشترین مقدار ممکن باشد.

همانطورکه متوجّه شده اید، این مسئله از نوع «زیرمجموعه» می باشد. زیرا که شما باید ببینید که هر شیء را باید بردارید یا برندارید.[1]

کاری که شما باید انجام دهید

شما می بایست توابع زیر را به ترتیب پُر کنید:

Class	Method
KnapsackUtility	**profitOfSelection**
KnapsackUtility	**weightOfSelection**
BlindAlgorithm	**isFeasible**
BlindAlgorithm	**findNextSubset**
BlindAlgorithm	**run**
GreedyAlgorithm	**Run**
Coordinator	**increaseSize**
Coordinator	**createRandomObjects**
Coordinator	**CreateRandomCapacity**

در هر یک از توابع سعی شده است به قدر کافی توضیحات لازمه در مورد ورودی وخروجی تابع و همچنین کاری که باید انجام شود به صورت comment داده شود.

[1] برای توضیحات بیشتر به فیلم شماره 4 (هفته ی چهارم) دقایق 25 الی 50 مراجعه کنید.

مسئله درخت جستجوی دودویی بهینه

صورت مسئله

مسئله ی درخت جستجوی دودویی بهینه (Optimum Binary Search Tree - OBST) بدین شرح است که تعدادی گره (node) داریم که هر کدام دارای یک کلید (key) و احتمال دسترسی به آن هستند. می خواهیم با این گره ها یک درخت جستجوی دودویی بسازیم. درخت جستجوی دودویی درختی است که هر گره آن حداکثر دو فرزند (فرزند چپ و فرزند راست) دارد. در عین حال، مقادیر کلید گره های موجود در زیردرخت سمت راست یک گره از مقدار کلید خود آن بیشتر، و مقادیر کلید گره های موجود در زیردرخت سمت چپ یک گره از مقدا کلید خود آن کمتر است. با توجّه به اینکه برای هر گره یک احتمال دسترسی تعریف شده است، مفهومی به عنوان «زمان دسترسی میانگین» (Average Access Time) برای درخت تعریف شده است که عبارتست از حاصل جمع ضرب احتمال دسترسی هر گره در عمق آن در درخت. هدف آن است که با این گره ها یک درخت جستجوی دودویی بسازیم که زمان دسترسی میانگین آن کمترین حدّ ممکن باشد.

با توجّه به اینکه ترتیب اضافه کردن گره ها به درخت جستجوی دودویی به صورت یکتا آن را تعیین می کند[۲]، مسئله معادل است با این موضوع که گره ها را به چه ترتیبی به درخت اضافه کنید. بنابراین، این مسئله از نوع «جایگشتی» می باشد؛ زیرا شما می بایست برای این گره ها یک جایگشت پیدا کنید[۳].

کاری که شما باید انجام دهید

شما می بایست توابع زیر را به ترتیب پُر کنید:

Class	Method
BSNode	**avgAccess**
BSTree	**addNode**
BSTree	**averageAccess**
BSTree	**createTree**
BSTree	**getNodesPermutation**
BlindAlgorithm	**permutation**
BlindAlgorithm	**run**
GreedyAlgorithm	**run**
Coordinator	**increaseSize**
Coordinator	**generateRandomNodes**

در هر یک از توابع سعی شده است به قدر کافی توضیحات لازمه در مورد ورودی وخروجی تابع و همچنین کاری که باید انجام شود به صورت comment داده شود.

[۲] هرچند، ممکن است یک درخت جستجوی دودویی از بیش از یک جایشگت حاصل شود.
[۳] برای توضیحات بیشتر به فیلم شماره پنج (هفته 5) دقایق 2 الی 50 مراجعه کنید.