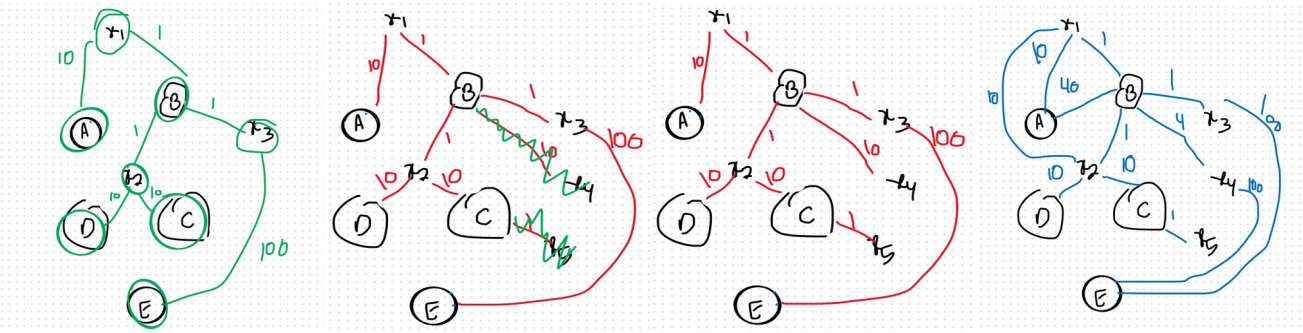
پروژه درس ساختمان داده ها
مسأله درخت اشتاینر

استاد کشتکاران

حسین خادمیان، دانشگاه شیراز

در ابتدا با مطالعه دو صفحه ابتدایی پروژه، شروع به حل مسأله به صورت ذهنی کردم.
راه حل ابتدایی خودم، مانند شیوه پیشنهادی طرح پروژه، تشکیل درخت پوشای مینیمال و سپس حذف رئوس بی کاربرد بود.
در (method1) یـا همـان راه حـل ابتـدایی، جهت پیاده‌سـازی این روش در بخش اول، از کروسـکال بـا WithPathComp UnionBySize پیشنهادی متن پروژه استفاده شده است.
سپس جهت حذف یال های اضافی، گراف/درخت پوشای مینیمال را تبدیل به یک درخت معمولی (با هر تعـداد فرزنـد و داشـتن والـد) میکنم. ریشـه این درخت یکی از رئوس ترمینال گراف خواهد بود.
پس از تشکیل درخت اقدام به حذف برگ‌های غیر پایانی این درخت می کنم. دلیل اینکار این است که، اگر راسی در مسیر رسیدن بـه یـک ترمینـال نباشد، پس قابلیت حذف دارد. وگرنه ، اگر در مسیر تولید یک ترمینال باشد، امکان حذف آن از درخت پوشای اولیه را ندارم چرا که بـا حـذف آن، تـک مسیر رسیدن به راس ترمینال از دست می رود. علت تک بودن مسیر هم تبدیل گراف به درختی بی دور در مرحله اول بوده است.
با حذف برگ ها، در هر مرحله برگ‌های جدیدی امکان تولید دارد. با تکرار مرحله قبل تمامی برگ‌های اضافی را حذف میکنم، در انتها که دیگر بـرگی اضافی باقی نمانده بود، درختی مینیمال شامل حداقل رئوس قابل تولید از درخت پوشای مینیمال مرحله قبل داریم.
در شکل‌های زیر به ترتیب از راست به چپ شیوه تبدیل این گراف به پاسخ رسم کرده ام.



اشکال این دو مرحله، حذف مسیر هایی از مسأله اس که در مرحله 2 امکان بهینه سازی بیشتری به مـا میـداد. کـه البتـه در method1 این مشـکل برطرف نشده است.
پیشنهاد من برای راه حل دوم: دو مرحله ابتدایی یکسان خواهد بود ولی به صورت Iterative در تمامی لبه هایی که شـامل ابتـدا 2 ترمینـال سـپس یک ترمینال و در آخر بی ترمینال است جستجو میکنم که آیا در گراف اولیه امکان تولید مسیری جایگزین بـا هزینـه کمـتر وجـود دارد یـا خـیر، و اگـر امکان آن وجود دارد نحوه مدیریت دور ها هم باید برسی شود و دور به نحوی حذف شود که هنوز پیاده‌سازی نکرده ام.

در ادامه گزارشی از این روند که منطبق بر push های گیت پروژه است ارایه می شود:

Initial commit :

در این مرحله، پروژه کلی بر مبنای JVM و زبان Kotlin تولید کردم، در این ساختار از سیستم بیلد Gradle استفاده شده است.

add proper graph input :

در ابتدا، دوست داشتم گراف و اطلاعات مسأله به صورت ساختار یافته و کلاس بندی شده جهت پردازش در برنامه داشته باشم.
بنابراین کلاس‌های Edge ،Vertex و Graph تعریف کردم.
در کلاس Vertex اطلاعات یک راس، که در ابتدای پروژه فقط شامل کلید معرف راس بود نگهداری می شود.

در کلاس Edge اطلاعات یک لبه، که شامل دو شی از کلاس راس و همچنین وزن این لبه بود نگه داری می شود.
در کلاس Graph اطلاعات مسأله یعنی همان لیست لبه ها، رئوس و رئوس ترمینال نگه داری می شود.
همچنین یکی از موارد مهم از دید من، داشتن دید کلی از اطلاعات مسأله جهت تولید راهکار مناسب میشود. بنابراین برای کلاس گراف نمایشی قابل فهم اطلاعات ارایه کردم.
متد هایی کمکی شامل:
inputGraph : جهت دریافت اطلاعات کل گراف از کاربر (لبه ها و ترمینال ها)
inputEdge: جهت دریافت اطلاعات یک لبه
inputVertexes: جهت دریافت لیستی از رئوس در یک خط
اضافه کردم. در این روش ورودی اطلاعات لیست رئوس به طور خودکار از لبه ها استخراج می شود. بنابراین در این گراف راسی که هیچ اتصالی نداشته باشد وجود ندارد.

dsl and struct :

با توجه به اینکه زبان برنامه نویسی کاتلین قابلیت اجرای کامل در ماشین مجازی جاوا دارد، اما قابلیت‌های بیشتری نسبت به جاوا ارایه می دهد.
یکی از قابلیت‌های قدرتمند آن قابلیت نوشتن DSL ها و بلاک های کد با اهداف مختلف است.
در این کامیت من دی اس ال هایی که امکان تعریف داده مسأله به صورت **توصیفی** میدهد را ارایه کردم به این صورت که در بلاک graph یک گراف تولید می‌شود که در آن با دستور e لبه ها افزوده شده و با t هم ترمینال ها مشخص می شود! مثلا:

```
val data = graph{ e( t(“a”) to v(“b”) w 2) }
```

در تک خط بالا یک گراف شامل دو راس a و b تعریف شده که یک لبه به وزن 2 بین آنهاست. همچنین راس a یک ترمینال است.

Method1

dsl clone

method1 progress

add simple tree

complete method 1 and refactor code

در این کامیت ها به پیاده‌سازی روش اول پرداخته ام. این شیوه شامل معرفی
Tree یا همان درخت ساده جهت پیاده‌سازی حذف برگ‌ها و امکان تبدیل گراف به درخت تعریف شده است. همچنین جهت داشتن نمای کلی از مراحل کار، نمایشی مناسب از درخت هم ارایه کرده ام.
متد kruskal در گراف ورودی شیوه مسأله کروسکال را پیاده‌سازی میکند. در این تابع از روش ارایه شده کلاس گراف ها به صورت مجموعه های جداگانه، طبق لبه های موجود با هم تریکب می‌شوند و گراف/درخت پوشای مینیمال نهایی را به خروجی می دهد.
تابع Graph.toTree هم قابلیت تبدیل یک گراف بی دور مسأله به درخت را فراهم میکند. این تابع جهت ریشه درخت از یکی از ترمینال ها استفاده می کند، دلیل این انتخاب این است که در ادامه به دنبال برگ‌ها می‌گردم و با رئوس فرزند دار تغییری ایجاد نمیکنم. با توجه به اینکه همه رئوس غیر ترمینال ، استعداد حذف شدن دارند، پس انتخاب مناسب برای این مرحله تعیین ریشه که دارای فرزند است به راسی که قابلیت حذف ندارد یا همان ترمینال است.
تابع Tree.write نمایشی مناسب از درخت در کنسول ارایه میدهد. (جهت امور دیباگ)
تابع Tree.delete امکان حذف یک درخت از والد را فراهم میکند. (فقط دو درخت را از هم جدا میکند) کاربرد جهت حذف برگ ها.

تا این کامیت متد 1 به صورت کامل پیاده‌سازی شده و به صورت کامل پاسخ را ارایه می دهد. (البته که بهینه ترین راه حل نیست)
اما طبق مطالعه ادامه پروژه متوجه شدم که باید از دیتاست PUC در متن پروژه استفاده کنم. بنابراین در ادامه به پیاده‌سازی متد های خواندن ورودی مسأله و خروجی مد نظر اقدام کردم:

PUC in/out :

تابع readGraph یک گراف را بر اساس استاندار فایل‌های STP موجود در PUC به صورت خروجی میدهد.

تابع writeGraph هم یک خروجی طبق فرمت مد نظر پروژه از نتیجه محاسبات روش اول ارایه میدهد.

برنامه کنسول به این صورت اپدیت شد که اگر کاربر ورودی خط فرمان به آن بدهداز آدرس آن پردازش را انجام می‌دهد در غیر این صورت آدرس فایل را ردکنسول ورودی از کاربر میگیرد.

random picking root :

یکی از شک‌های مد نظرم جهت تست کردن، امکان متفاوت بودن عمل‌کرد شیوه مرحله 2 درصورت متفاوت بودن انتخاب ریشه بود، که با انتخاب رندوم یکی از ترمینال ها و مقایسه خروجی ها، شک برطرف شده و مطمئن شدم که درخت یونیک پوشای خروجی مرحله اول، درصورت ریشه بودن یکی از ترمینال ها حتمی یک خروجی منحصر به فرد دارد.

به دلیل تکه‌تکه خواندن پروژه، و حل تکه‌های مختلف مسأله، در انتها متوجه محدودیت زبان به جاوا و ... شدم، باتوجه به تشابه بسیار زیاد کاتلین و جاوا و قابلیت استفاده از کدها و برنامه‌های کاتلین در محیط جاوا، خواهشمند است این کد ها را بپذیرید.

در انتهای پروژه با توجه به لیست بنچمارک لینک ارایه شده و سورس دیتاست ورودی، تمام خروجی ها را محاسبه و در جدول زیر تکمیل کردم.

به علت تعداد زیاد پروژه های پایانی سال درصورت امکان حتمی در روش 2 که مرحله 3 اختیاری را تکمیل می‌کند، پیاده‌سازی میکنم.

با تشکر

Name	|V|	|E|	|T|	DC	Opt	Weight S1 Kruskal	Weight S2 Drop	Weight S3 Optimise
bip42p	1200	3982	200	NP?	**24657**	123100	36253	
bip42u	1200	3982	200	NP?	**236**	1199	308	
bip52p	2200	7997	200	NP?	*24526*	225722	38471	
bip52u	2200	7997	200	NP?	*234*	2199	323	
bip62p	1200	10002	200	NP?	*22843*	121697	36771	
bip62u	1200	10002	200	??	*219*	1199	269	
bipa2p	3300	18073	300	??	*35326*	336807	56993	
bipa2u	3300	18073	300	??	*338*	3299	433	
bipe2p	550	5013	50	NP?	**5616**	55711	9292	
bipe2u	550	5013	50	NP?	**54**	549	61	
cc10-2p	1024	5120	135	??	*35297*	117644	65638	
cc10-2u	1024	5120	135	??	*342*	1158	461	
cc11-2p	2048	11263	244	??	*63491*	232532	123883	
cc11-2u	2048	11263	244	??	*612*	2291	852	
cc12-2p	4096	24574	473	??	*121106*	463300	240160	
cc12-2u	4096	24574	473	??	*1172*	4568	1639	
cc3-10p	1000	13500	50	??	*12772*	105980	29448	
cc3-10u	1000	13500	50	??	*125*	1049	149	
cc3-11p	1331	19965	61	??	*15582*	140498	34385	
cc3-11u	1331	19965	61	??	*153*	1391	182	
cc3-12p	1728	28512	74	??	*18826*	181894	44571	
cc3-12u	1728	28512	74	??	*185*	1801	220	
cc3-4p	64	288	8	NP?	**2338**	7215	3644	
cc3-4u	64	288	8	NP?	**23**	71	24	
cc3-5p	125	750	13	NP?	**3661**	13875	7272	
cc3-5u	125	750	13	??	**36**	137	38	
cc5-3p	243	1215	27	??	**7299**	27322	14383	
cc5-3u	243	1215	27	??	**71**	269	87	
cc6-2p	64	192	12	NPm	**3271**	7690	4988	
cc6-2u	64	192	12	NPm	**32**	75	37	
cc6-3p	729	4368	76	??	**20270**	81601	41119	
cc6-3u	729	4368	76	??	**197**	804	249	
cc7-3p	2187	15308	222	??	*56799*	243926	119639	
cc7-3u	2187	15308	222	??	*549*	2408	739	
cc9-2p	512	2304	64	??	*17199*	58470	32532	
cc9-2u	512	2304	64	??	**167**	575	218	
hc10p	1024	5120	512	??	*59797*	103103	83897	
hc10u	1024	5120	512	??	*575*	1023	767	
hc11p	2048	11264	1024	??	*119492*	206056	167002	
hc11u	2048	11264	1024	??	*1145*	2047	1535	
hc12p	4096	24576	2048	??	*236949*	411938	334770	
hc12u	4096	24576	2048	??	*2262*	4095	3071	
hc6p	64	192	32	NPm	**4003**	6391	5279	
hc6u	64	192	32	NPm	**39**	63	47	
hc7p	128	448	64	NP?	**7905**	12865	10215	
hc7u	128	448	64	NP?	**77**	127	95	
hc8p	256	1024	128	NP?	**15322**	25774	20907	
hc8u	256	1024	128	NP?	**148**	255	191	
hc9p	512	2304	256	NP?	*30242*	51593	41777	
hc9u	512	2304	256	NP?	*292*	511	383	