

پروژه درس ساختمان داده ها

مسئله درخت اشتاینر

استاد کشتکاران

حسین خادهمیان، دانشگاه شیراز

در ابتدا با مطالعه دو صفحه ابتدایی پروژه، شروع به حل مسئله به صورت ذهنی کردم.

راه حل ابتدایی خودم، مانند شیوه پیشنهادی طرح پروژه، تشکیل درخت پوشای مینیمال و سپس حذف رئوس بی کاربرد بود.

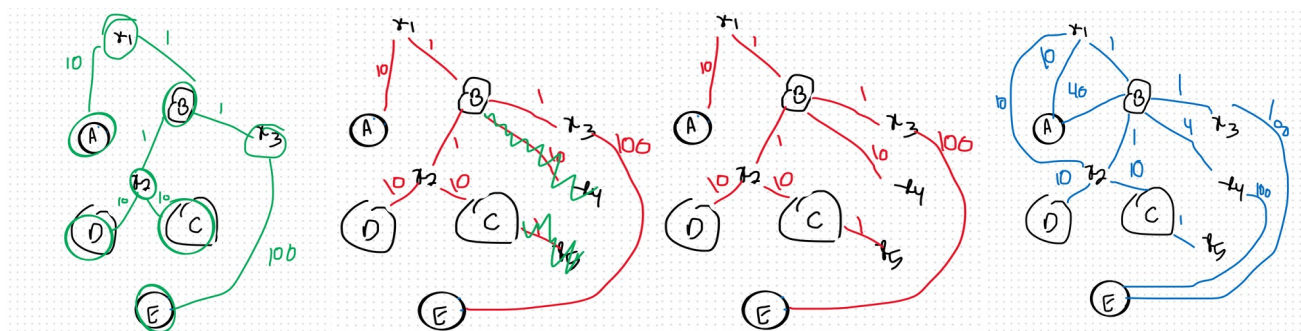
در (method1) یا همان راه حل ابتدایی، جهت پیاده سازی این روش در بخش اول، از کروسکال با UnionBySize WithPathComp پیشنهادی متن پروژه استفاده شده است.

سپس جهت حذف یال های اضافی، گراف/درخت پوشای مینیمال را تبدیل به یک درخت معمولی (با هر تعداد فرزند و داشتن والد) میکنم. ریشه این درخت یکی از رئوس ترمینال گراف خواهد بود.

پس از تشکیل درخت اقدام به حذف برگ های غیر پایانی این درخت می کنم. دلیل اینکار این است که، اگر راسی در مسیر رسیدن به یک ترمینال نباشد، پس قابلیت حذف دارد. وگرنه، اگر در مسیر تولید یک ترمینال باشد، امکان حذف آن از درخت پوشای اولیه را ندارم چرا که با حذف آن، تک 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 برگ های جدیدی امکان تولید دارد. با تکرار مرحله قبل تمامی برگ های اضافی را حذف میکنم، در انتها که دیگر برگی اضافی باقی نمانده بود، درختی مینیمال شامل حداقل رئوس قابل تولید از درخت پوشای مینیمال مرحله قبل داریم.

در شکل های زیر به ترتیب از راست به چپ شیوه تبدیل این گراف به پاسخ رسم کرده ام.



اشکال این دو مرحله، حذف مسیر هایی از مسئله اس که در مرحله 2 امکان بهینه سازی بیشتری به ما میداد. که البته در method1 این مشکل برطرف نشده است.

پیشنهاد من برای حل دوم: دو مرحله ابتدایی یکسان خواهد بود ولی به صورت Iterative در تمامی لبه هایی که شامل ابتدا 2 ترمینال سپس یک ترمینال و در آخر بی ترمینال است جستجو میکنم که آیا در گراف اولیه امکان تولید مسیری جایگزین با هزینه کمتر وجود دارد یا خیر، و اگر امکان آن وجود دارد نحوه مدیریت دور ها هم باید بررسی شود و دور به نحوی حذف شود که هنوز پیاده سازی نکرده ام.

در ادامه گزارشی از این روند که منطبق بر push های گیت پروژه است ارائه 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 مسئله به صورت ساختار یافته و کلاس بندی شده جهت پردازش در برنامه داشته باشم. بنابراین کلاس های Vertex، Edge 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 قابلیت های بیشتری نسبت به جاوا ارایه می دهد.

یکی از قابلیت های 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 می شود که در آن با دستور e لبه ها افزوده شده و با t هم ترمینال ها مشخص می شود! مثلاً:

```
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 پیاده سازی روش اول پرداخته ام. این شیوه شامل معرفی Tree یا همان درخت ساده جهت پیاده سازی حذف برگ ها 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 پیاده سازی میکند. در این تابع از روش ارایه شده کلاس گراف ها به صورت مجموعه های جداگانه، طبق لبه های موجود با هم ترکیب می شوند 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 برگ ها می گردم 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 پیاده سازی 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 پیاده سازی متد های خواندن ورودی و مسأله و خروجی مد نظر اقدام کردم:

: PUC in/out

تابع readGraph یک گراف را بر اساس استاندارد فایل های 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 بدهد از آدرس آن پردازش را انجام می دهد در غیر این صورت آدرس فایل را ردکنسول ورودی از کاربر میگیرد.

: random picking root

یکی از شک های مد نظرم جهت تست کردن، امکان متفاوت بودن عمل کرد شیوه مرحله 2 در صورت 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، در صورت ریشه بودن یکی از ترمینال ها حتمی یک خروجی منحصر به فرد دارد.

به دلیل تکه تکه خواندن پروژه، و حل تکه های 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 برنامه های کاتلین در محیط جاوا، خواهشمند است این کد ها را بپذیرید.

به علت تعداد زیاد پروژه های پایانی سال در صورت امکان حتمی در روش 2 که مرحله 3 اختیاری را تکمیل می کند، پیاده سازی میکنم.
با تشکر