در این تمرین کامپیوتری هدف این است که شما با زبان برنامه­نویسی Solidity و همچنین یک API[[1]](#footnote-1) به نام web3.js آشنا شوید به طوریکه بتوانید یک نرم­افزار غیرمتمرکز[[2]](#footnote-2) ساده را پیاده سازی کنید. در نرم­افزار­های غیرمتمرکز نیز همانند سایر نرم­افزارها لازم است یک قسمت سمت کاربر[[3]](#footnote-3) و یک قسمت سمت سرور[[4]](#footnote-4) پیاده سازی شود، که در این تمرین شما با پیاده سازی هر دو سمت آشنا می­شوید.

1. دفترچه­حساب زنجیره­ی بلوکی[[5]](#footnote-5)

هدف این است که یک سیستم غیرمتمرکز طراحی شود که بتواند بدهی­ و طلب افراد را ثبت کند. درواقع نمونه­ی غیرمتمرکز از یک دفترچه­ی حساب. برای شفاف شدن بیشتر، یک مثال از نحوه­ی کار این سیستم در ادامه آورده می­شود.

فرض کنید که علیرضا و الهام و شبنم سه دوست هستند که معمولا با یکدیگر بیرون می­روند و غذا میخورند. آخرین بار که علیرضا و الهام با یکدیگر ناهار خورده اند، هزینه آن را الهام پرداخت کرده است. بنابراین علیرضا 30 هزار تومان به الهام بدهی دارد. همچنین یک بار که الهام و شبنم برای شام بیرون رفته­اند شبنم هزینه آن را پرداخت کرده است، پس الهام 30 هزار تومان به شبنم بدهکار است.

حال فرض کنید که شبنم به پول نیاز دارد و مبلغ 30 هزار تومان از علیرضا قرض بگیرد. در این لحظه هر سه نفر می­توانند توافق کنند که هیچ یک از آن­ها به دیگری بدهی­ای ندارد و از تبادل پول خودداری کنند. درواقع هر زمانی که یک دور[[6]](#footnote-6) از بدهی­ها وجود داشته باشد می توان آن دور را حذف کرد و از جابجا شدن پول جلوگیری کرد.

ما قرار است که یک سیستم غیرمتمرکز طراحی و پیاده سازی کنیم که مشخص کند هر شخص به چه کسی و چه مبلغی بدهکار است، بدون اینکه نیاز باشد یک اورگان سومی[[7]](#footnote-7) در میان باشد. به صورتی­که مثلا وقتی الهام هزینه ناهار علیرضا را پرداخت می­کند از او بخواهد که در شبکه زنجیره­ی بلوک بدهکار بودنش را تصدیق کند. فضای موجود و عمومی و همچنین قابل دسترسِ شبکه­ی زنجیره­ی بلوک می­تواند به عنوان محل ذخیره سازی و یا به عبارتی سمت سرور برای لیست بدهی های هر شخص استفاده شود. برای راحتی کار با این سرور لازم است که یک محیط کاربری نیز طراحی شود که جلوتر انجام خواهید داد. ☺

1. شروع
2. نصب نرم­افزارهای پیشنیاز: لازم است که بر روی سیستم خود Node.js را نصب کنید. می توانید از لینک زیر کمک بگیرید.
3. نصب شبیه ساز شبکه اتریوم. برای اینکار از دستور npm install –g ganache-cli استفاده کنید. با این کار یک سرور به نام ganache بر روی سیستم شما نصب می شود که با اجرا شدن آن تعدادی اکانت اتریوم با موجودی های تعیین شده ایجاد می شود که از آن­ها می­توانید برای تمرین کامپیوتری استفاده کنید. برای اجرا شدن این سرور دستور ganache-cli را وارد کنید. برای متوقف کردن سرور نیز می­توانید از Ctrl+C استفاده کنید.
4. از صفحه­ی درس کد starter را دانلود کنید.
5. در جستجوگر وب خود صفحه­ی مقابل را باز کنید. <https://remix.ethereum.org> در قسمت ‘Run’ محیط اجرایی[[8]](#footnote-8) را بر روی ‘Web3 Provider’ تنظیم کنید. و آدرس آن را نیز بر روی <http://localhost:8545> که آدرس پیشفرض نیز هست، تنظیم کنید. محیطی که در آن هستید جایی است که لازم است شما قرارداد هوشمند[[9]](#footnote-9) خود را بنویسید و آن را کامپایل کنید.
6. در پوشه ای که از سایت درس دانلود کرده اید یک فایل به نام index.html وجود دارد. آن را در مرورگر خود باز کنید. باید صفحه ای را تحت عنوان ‘Blockchain Splitwise’ بتوانید ببینید. توصیه می­شود که قسمت javascript console مرورگر خود را نیز باز کنید و خطاهای احتمالی را مشاهده کنید. ( از این لینک نیز می­توانید کمک بگیرید) در صورتیکه همه چیز به خوبی پیش رفته باشد. شما نباید خطایی را مشاهده کنید. البته احتمالا هشدارهایی را می­بینید که می­توانید آن­ها را نادیده بگیرید.
7. فایل­های درون پوشه­ی دانلود شده را در محیط برنامه نویسی مورد علاقه خود باز کنید. (مانند Atom, Sublime,.. ) شما برای کامل شدن سمت کاربر لازم است که فایل script.js را کامل کنید. توصیه می­شود که سایر فایل های موجود را نیز بررسی کنید.
8. قبل از شروع توصیه می­شود که راجع به web3.js و برنامه نویسی solidity اطلاعات بیشتری کسب کنید. همچنین فکر کردن راجع به نحوه­ی کار سیستم کلی و همچنین نحوه ی ذخیره سازی اطلاعات، قبل از کدزنی می­تواند مفید باشد.
9. در قسمت بعد کارهایی که قرار است انجام دهید با جزئیات بیشتری توضیح داده می­شود، بعد از اینکه قرارداد هوشمند خود را نوشتید و آن را بر روی شبکه ثبت کردید لازم است که آدرس[[10]](#footnote-10) آن و همچنین چکیده[[11]](#footnote-11) آن را در فایل script.js به­روزرسانی کنید. چکیده را می­توانید از قسمت ‘Compile’ و آدرس را نیز می­توانید از قسمت ‘Deployed Contracts’ در بخش ‘Run’ بیابید و کپی کنید.
10. نیازمندی­ها

همانطور که قبلا توضیح داده شد، این تمرین کامپیوتری از دو بخش مهم تشکیل می­شود: قرارداد هوشمند که به زبان solidity نوشته می­شود و در شبکه­ی آنلاین زنجیره­ی بلوک ثبت می­شود و قسمت سرور شما را تشکیل می­دهد و همچنین یک محیط کاربری که به زبان JavaScript نوشته می­شود و درمرورگر شما اجرا می­شود و از طریق web3.js به سرور شما وصل می­شود و اطلاعاتی را که میخواهید نمایش می­دهد.

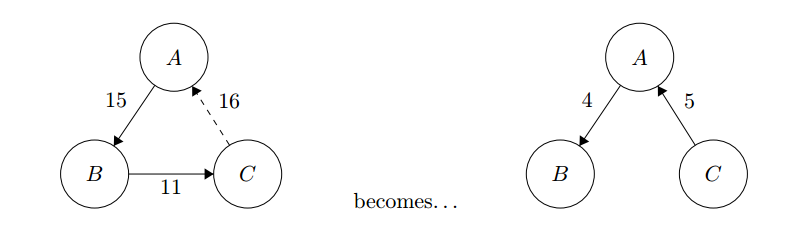
1. توابعی سمت کاربر
2. getUsers() : این تابع یک لیست از آدرس­های شبکه­ی اتریوم را برمی­گرداند. شما می­توانید آدرس تمام افراد موجود در شبکه را برگردانید. و یا اینکه فقط افرادی را برگردانید که تا به حال فعالیتی مالی­ای داشته­اند. این تابع می­تواند به عنوان تابع کمکی در دیگر توابع مورد استفاده قرار بگیرد.
3. getTotalOwed(user) : این تابع آدرس یک کاربر را می­گیرد و کل مبلغی را که آن کاربر بدهکار است برمی­گرداند.
4. getLastActive(user) : این تابع زمان آخرین فعالیت کاربر را برمی­گرداند. منظور از آخرین فرستادن IOU و یا ذکر شدن به عنوان بستانکار در یک IOU توسط یک کاربر دیگر است. فرمت زمانی باید به صورت ثانیه­های گذشته از مبدا تاریخ ( 1 ژانویه 1970) باشد.
5. add\_IOU(creditor, amount) : این تابع به عنوان تاییدیه فرستاده می­شود که نشان می­دهد شخصی که آن را صدا می­زند مبلغی معادل amount را به فرد creditor بدهکار است. این تابع چیزی را برنمی­گرداند. در این تابع لازم است که مشکل وجود حلقه نیز برطرف شود که جلوتر در مورد آن توضیح خواهیم داد.
6. توابع سمت سرور
7. lookup(address debtor, address creditor) public view returns (unit32 ret) : این تابع مبلغی را که فرد debtor به شخص creditor بدهکار است، برمی­گرداند. برای این کار کافی است که در بدنه ی تابع متغیر ret را مقداردهی کنید. و نیازی به نوشتن return در انتهای تابع نیست.
8. add\_IOU(address creditor, uint32 amount, …) : این تابع تاییدیه نهایی در سرور را انجام می­دهد و نشان می­دهد که msg.sender مبلغ amount را به فرد creditor بدهکار است. و اگر شخص بدهکار از قبل بدهکار بوده است طبیعتا مبلغ جدید با مقدار قبلی جمع می­شود. amount لازم است که عدد مثبتی باشد. شما می­توانید بسته به نیازتان آرگومان های دیگری برای این تابع تعریف کنید.

شما می­توانید بسته به نیازتان توابع دلخواه دیگری را در هریک از دوسمت کاربری و سرور تعریف کنید و از آن­ها استفاده کنید. در سمت کاربری می­توانید توابع سمت سرور را به صورت BlockchainSplitwise.functionname(arguments) صدا بزنید.

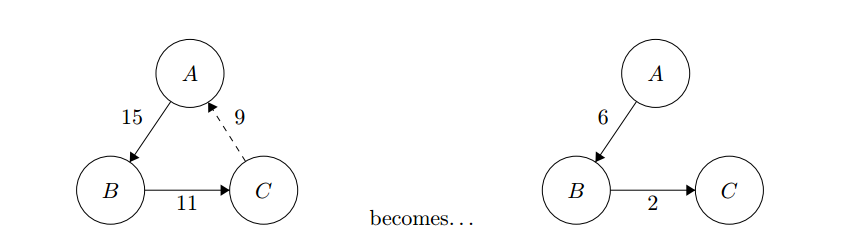
1. حل مشکل وجود حلقه­ی بدهی­ها

برای تحلیل راحت­تر، از مدل گراف جهت­دار وزندار برای مسئله­ی خود استفاده می­کنیم، بدین صورت که هر یک از کاربرها یک گره محسوب می­شوند و هریک از یال­ها وزنی دارد که مقدار وزن آن نشان­دهنده­ی مبلغ بدهی است. جهت یال ها از بدهکار به سمت بستانکار است. مثلا وقتی که شخص A مبلغ X را به شخص B بدهکار است آنگاه: . نرم­افزار ما لازم است به­گونه­ای کار کند که اگر حلقه­ای ایجاد شد آن را از بین ببرد برای این­کار لازم است که کوچکترین یال در حلقه پیدا شود و مقدار آن از تک­تک یال­ها کم شود به صورتیکه یک یکی از یال­ها به ارزش صفر برسد.

به عنوان مثال اگر و آنگاه اگر C بخواهد تراکنش را اضافه کند لازم است که نرم­افزار ما گراف را به صورت زیر تغییر دهد.



*به عنوان یک مثال دیگر اگر C بخواهد تراکنش را اضافه کند آنگاه گراف جدید به صورت زیر خواهد بود.*



لازمه­ی انجام این­کار این است که شما در تابع add\_IOU در قسمت کاربری قبل از فرستادن دستور به سمت سرور چک کنید که آیا حلقه­ای وجود دارد یا نه و درصورت وجود حداقل یکی از آن­ها را حذف کنید. نگران موارد پیچیده نباشید و یا اینکه لازم نیست به دنبال پیدا کردن راه­های بهینه برای حذف حلقه باشید. بنابراین لازم است در توابع سمت سرور این فرض را درنظر بگیرید که هیچ حلقه­ای وجود ندارد و قسمت حذف حلقه در سمت کاربر انجام می­گیرید.

در کدهای سمت کاربر یک تابع کمکی به نام doBFS(start, end, getNeigbors) برای شما قرار داده شده است که الگوریتم bread-first search را انجام می­دهد و خروجی آن لیستی از گره­هاای است که گره start را به گره end وصل می­کنند، البته درصورتیکه مسیری بین این دو گره وجود داشته باشد. دقت شود که آرگومان سوم آن خود یک تابع است که یک گره را می­گیرد و همسایه­های آن را به صورت یک لیست برمی­گرداند و لازم است خودتان آن را بنویسید. در صورتیکه مسیری بین دوگره وجود نداشته باشد، خروجی تابع یک لیست خالی خواهد بود. استفاده از این تابع اختیاری است و شما می­توانید از هر الگوریتم دیگری برای پیدا کردن مسیر استفاده کنید.

آخرین چیزی که به نظر می­رسد باید انجام دهیم، پرداخت بدهی است به این صورت که فرض کنید آخرین بار علیرضا مبلغ 40 هزار تومان به شبنم بدهکار است حال اگر پول خود را به شبنم پرداخت کند لازم است که شبنم در شبکه یک تاییدیه بفرستد که مبلغ 40 هزار تومان به علیرضا بدهکار است با این کار یک حلقه­ی کوچک به وجود می­آید که انتظار می­رود نرم­افزار ما آن را تشخیص دهد و حذف کند و نتیجه­ی آن این باشد که هیچ یک از علیرضا و شبنم به دیگری بدهکار نیست. در واقع برای این حالت خاص لازم نیست که شما کار خاصی را انجام دهید، درصورتیکه قسمت قبل را به خوبی پیاده­سازی کرده باشید، این حالت نیز خودبه­خود انجام می­شود.

1. توصیه های نهایی

دست شما باز است که هر طور که میخواهید قراداد هوشمند خود را بنویسید فقط لازم است که توابع خواسته شده را حتما پیاده­سازی کنید. همچنین سعی کنید قرارداد هوشمند شما کمترین میزان حافظه و کمترین میزان انجام محاسبات را به خود اختصاص دهد، چرا که اینکار باعث می­شود که استفاده از قرارداد شما ارزان­تر تمام شود.

توصیه می­شود که از بدهی منفی حتما خودداری کنید زیرا این­کار پیچیدگی را افزایش می­دهد.

1. Application Protocol Interface [↑](#footnote-ref-1)
2. Decentralized Application(DApp) [↑](#footnote-ref-2)
3. Client side [↑](#footnote-ref-3)
4. Server side [↑](#footnote-ref-4)
5. Blockchain Splitwise [↑](#footnote-ref-5)
6. Cycle [↑](#footnote-ref-6)
7. Third party [↑](#footnote-ref-7)
8. Environment [↑](#footnote-ref-8)
9. Smart contract [↑](#footnote-ref-9)
10. Contract address [↑](#footnote-ref-10)
11. ABI [↑](#footnote-ref-11)