بسم الله الرحمن الرحیم

استاد: دکتر محمدعلی مداح علی

درس: بلاکچین

تمرین: عملی سری 1 بلاکچین

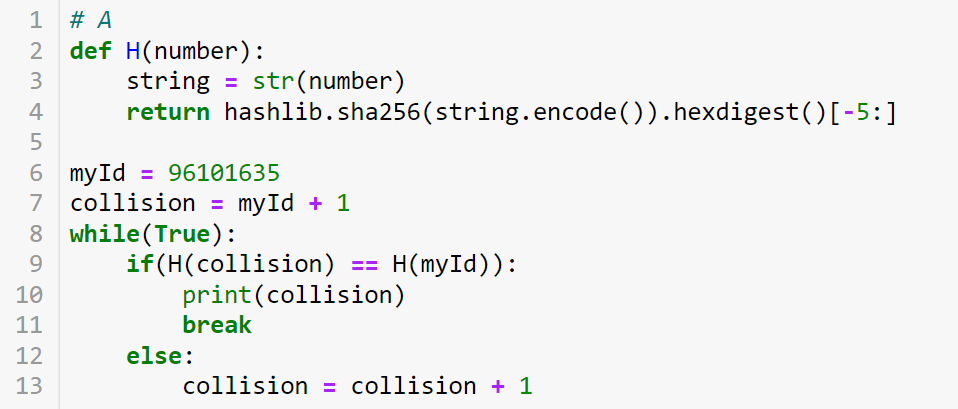
دانشجو: امیرحسین رستمی 96101635

پاییزِ 99

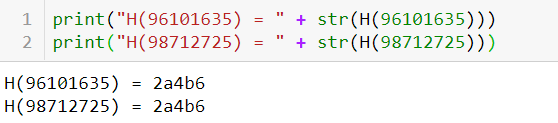
سوال اول:

a.کد نوشته شده برای این قسمت به شرح زیر است:

ابتدا collision = myId + 1 شروع می کنیم و در هر مرحله از آن هش میگیریم و با هشِ خود myId چک می کنیم و در صورت برابری متوقف شده و نتیجه را اعلام می کنیم.(رویکردی شبیه به brute force) و در صورت عدم برابری یک واحد به collision اضافه می کنیم و این حلقه را آنقدر تکرار می کنیم تا سرانجام به پاسخ برسیم.



پس از اجرای کد فوق دریافتیم که در collision = 98712725،برابریِ هش ها اتفاق می افتد و داریم که هش این عدد و شماره دانشجویی بنده یکسان است.



b.همانطور که از مساله birthday problem اطلاع دارید اگر به صورت رندوم ۲۳ نفر را انتخاب کنیم با احتمال بالای ۵۰ درصد دو نفر در این جمع هستند که تاریخ تولدشان با هم برابر است! حال فرض کنید که تابع هش ای داریم که فضای حالت خروجی هایش ۳۶۵ حالت باشد،birthday problem به ما می گوید که اگر دسته های ۲۳ تایی از اعداد را برداریم با احتمال بالای ۵۰ درصد دو عدد در این دسته هستند که دارای hash عه برابر اند! از همین نکته استفاده می کنیم و به جای رویکرد brute force ای،دسته های n تایی از اعداد را برمیداریم که می دانیم با توجه به فضای حالت hash ِ جدید ما با احتمال خوبی collision در این جمع رخ می دهد و آنقدر حلقه را تکرار می کنیم تا سرانجام به collision برسیم.حال می خواهم در ادامه نحوه به دست آوردن این n را توضیح بدهم.

فرض کنید که تعداد حالت hash ما D باشد(که در این سوال ).

به دنبال محاسبه عبارت E[number of collisions] هستیم،در یک دسته k-1 تایی از اعداد،احتمال اینکه عدد بعدی(عدد k ام) با هیچ کدام از k-1 اعضای دسته collision نداشته باشد برابر است با:

بنابراین احتمال اینکه حداقل با یک نفر از k-1 عضو دسته تلاقی داشته باشد(Hash اش با آن برابر باشد)واضحا برابر است با:

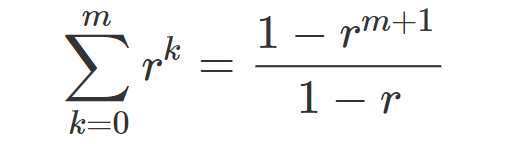
برای محاسبه E[number of collisions] در یک دسته ی n تایی نیاز است تا موارد زیر را محاسبه کنیم،(ابتدا دسته را خالی فرض کنید و سپس پله پله اعداد را به آن اضافه می کنیم):

* تعداد تلاقی های رخ داده در اثر اضافه کردن عدد اول.
* تعداد تلاقی های رخ داده در اثر اضافه کردن عدد دوم.
* ...
* تعداد تلاقی های رخ داده در اثر اضافه کردن عدد n ام.

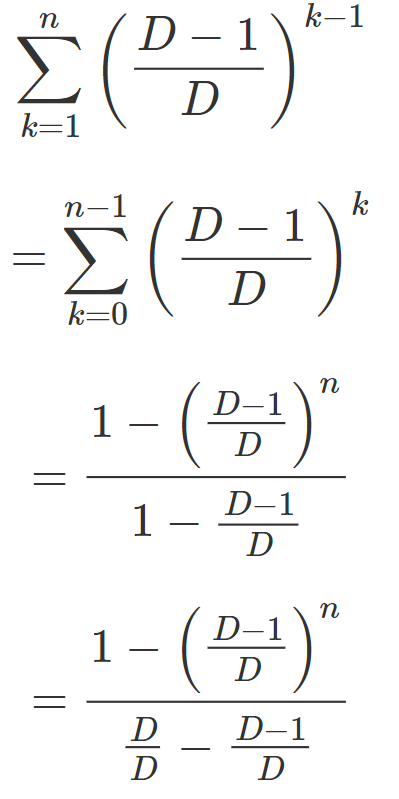
لذا داریم که:

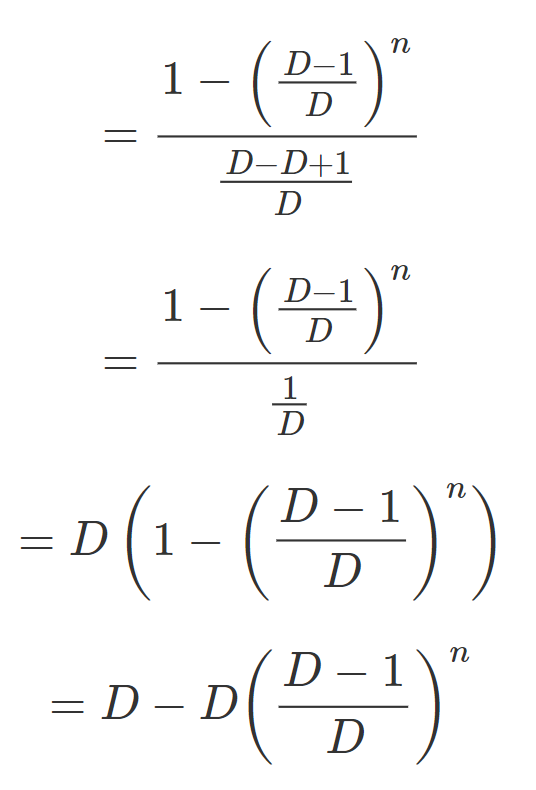
ادامه در صفحه بعد.

به کمک رابطه زیر به ساده تر کردن سیگمایِ به دست آمده می پردازیم:



حال داریم که:



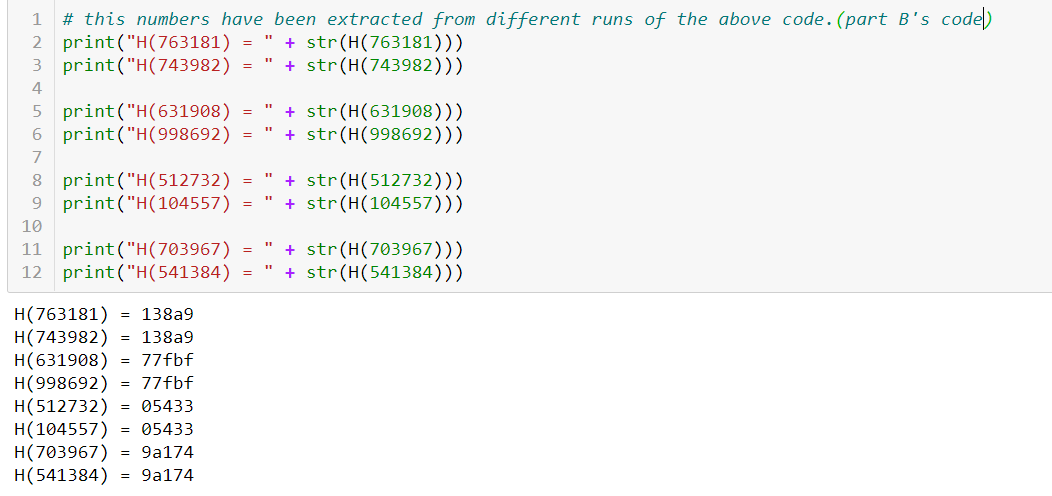


ادامه در صفحه بعد.

بنابراین داریم که E[number of collision] برابر عبارت زیر می گردد:

حال ما در کد آن n ای که باعث میشود عبارت فوق بزرگتر مساوی یک شود را درنظر میگیریم و سپس دسته های n تایی از اعداد برمیدارم و به یافتن collision می پردازیم:

خروجی چند نمونه از ران های مختلف کد را نشان می دهیم:)4 جفت collision حاصل از 4 بار ران مختلف کد)



c.ابتدا ایده حل سوال را بیان می کنیم،می دانیم که هم سمت فرستنده و هم سمت گیرنده secret key را دارند و می دانیم هم اینکه کسی اصل پیام را بشنود مشکلی نیست و برایِ ما امضا شدن پیام مهم است،لذا چنین عمل می کنیم:

سمت فرستنده:

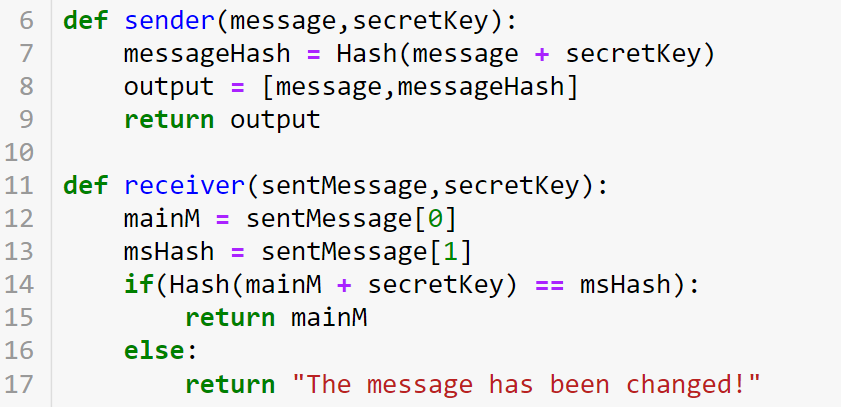
1. ابتدای پیام اصلی را با secret key،الحاق(concat) می کنیم.
2. سپس Hash پیام فوق را محاسبه می کنیم.
3. و به گیرنده [mainMessage,aboveHash] را ارسال می کنیم.

سمت گیرنده:

1. mainMessage دریافت شده را با secret key ای که فقط من و فرستنده داریم الحاق(concat) می کنیم.
2. Hash ِ رشته به دست آمده از مرحله قبل را محاسبه می کنیم:
   1. اگر با aboveHash(تیکه یِ دوم پیغام دریافت شده از سمت فرستنده) برابر بود-با توجه به مقاوم بودن تابع hash در محاسبه وارون-اطمینان پیدا می کنیم که پیغام از سمت همان کسی است که secret key را دارد و تمام.
   2. اما اگر برابر نبود یعنی یا پیغام دستکاری شده است یا امضایِ شخص مورد تایید نیست که در هردو حالت خروجی “The message has been changed!” را می دهد.

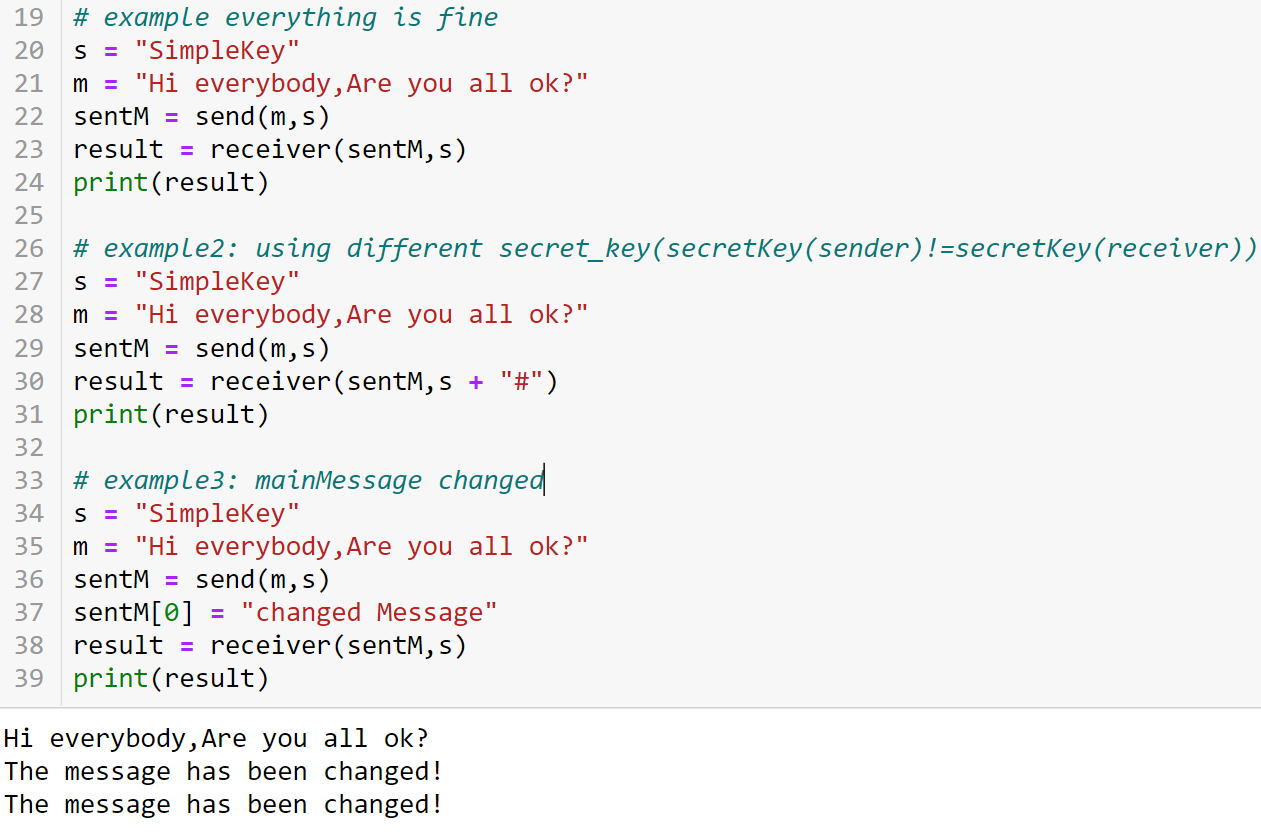
با توجه به مقاوم بودن تابع Hash در محاسبه ی وارون،سناریویِ فوق مطالبات مارا ارضا خواهد کرد.

کد سمت فرستنده و گیرنده به شرح زیر است:



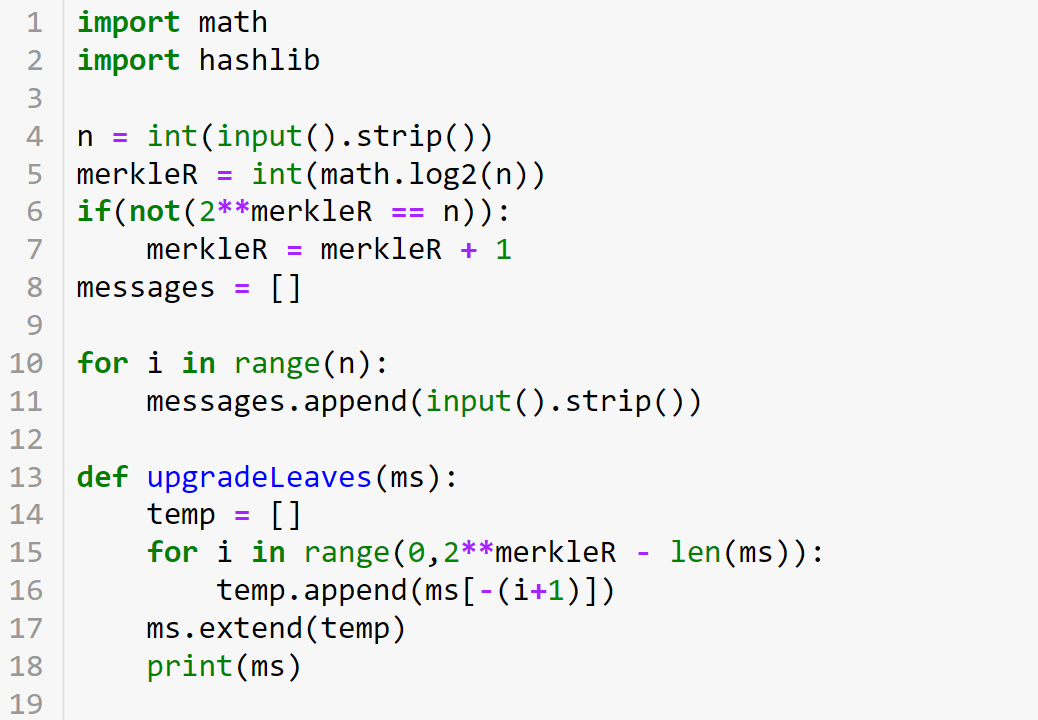
همانطور که مشاهده می کنید هر دو پیاده سازی sender و receiver طبق مطالب گفته شده انجام شده است و در صفحه بعد نیز چند نمونه از تست کیس های این بخش ضمیمه شده است.

3 نمونه تست کیس:



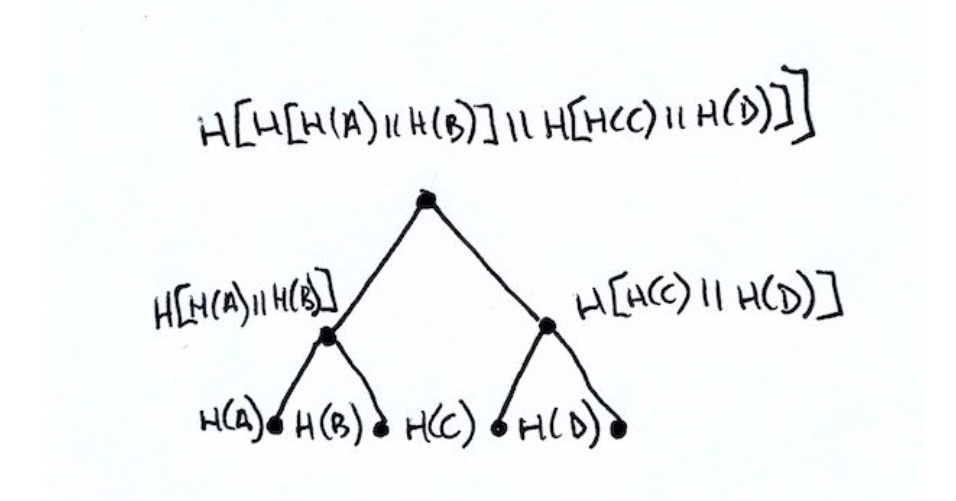
سوال دوم:

ابتدا با قطعه کد زیر،تعداد برگ های درخت را به صورت توانی از ۲ در می آوریم:

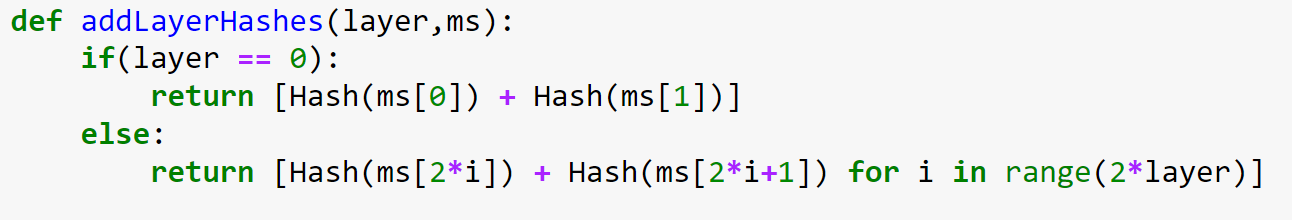


تابع upgradeLeaver،برگ هارا گسترش می دهد تا به صورت توانی از 2 در بیایند.

جهت محاسبه ی merkle tree به تصویر زیر توجه کنید:(نماد || به معنی الحاق یا همان concat است).

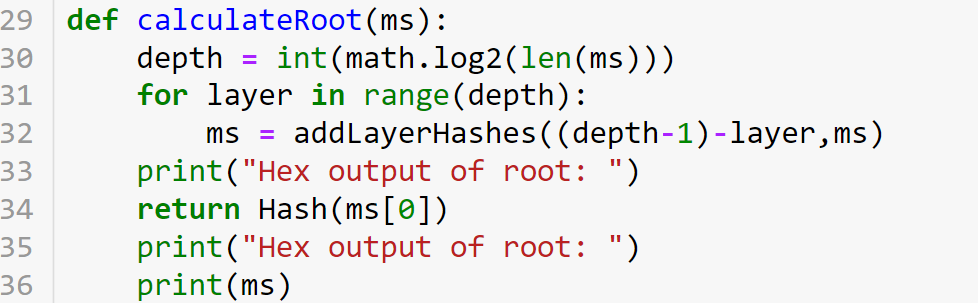


در کد برای محاسبه لایه لایه هش ها مانند تصویر فوق،از تابع addLayerHashes استفاده می کنم که در هر لایه،برای هر نود،به کمک اطلاعات فرزندانش،مقدارش محاسبه می گردد(مشابه تصویر فوق)



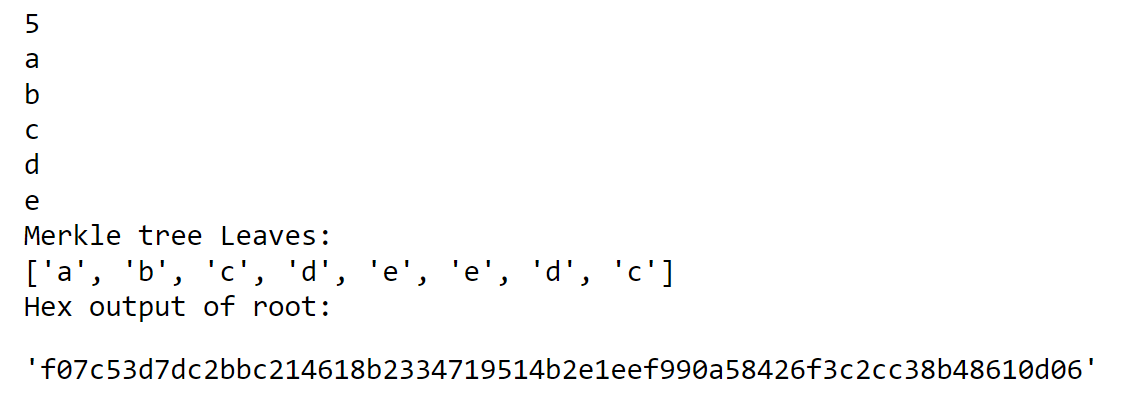
توجه کنید که تابع فوق شماره لایه را میگیرد و مقادیر نود های لایه با شماره layer-1 را به کمک مقادیر لایه با شماره layer که همان(ms) است محاسبه می کند و برمیگرداند.

حال به کمک تابع فوق و قطعه کد زیر merkle root که همان مقدار نود در لایه 0 است را محاسبه می کنیم.



مثال:

درخت زیر را درنظر بگیرید:(همان مثال صورت سوال،فرض کنید می خواهیم برای فایل هایِ a,b,c,d,e درختِ merkle طراحی کنیم،ابتدا n = 5 و سپس به ترتیب 5 فایل مدنظرمان را وارد می کنیم،پس از دریافت 5 عدد ورودی،برگ های upgrade شده(که تعدادشان به صورت توانی از 2 است) را محاسبه می کنیم و نشان می دهیم و سپس از لایه آخر شروع کرده و لایه لایه درخت merkle را تولید می کنیم و بالا می رویم و در نهایت مقدارِ Hex عه root ِدرخت merkle را در خروجی می نویسیم:



اگر هم به صورت دستی هش هارا حساب کرده و concat کنید به مقداری برابر با مقدار فوق برای ریشه درخت merkle خواهید رسید.

سوال سوم:

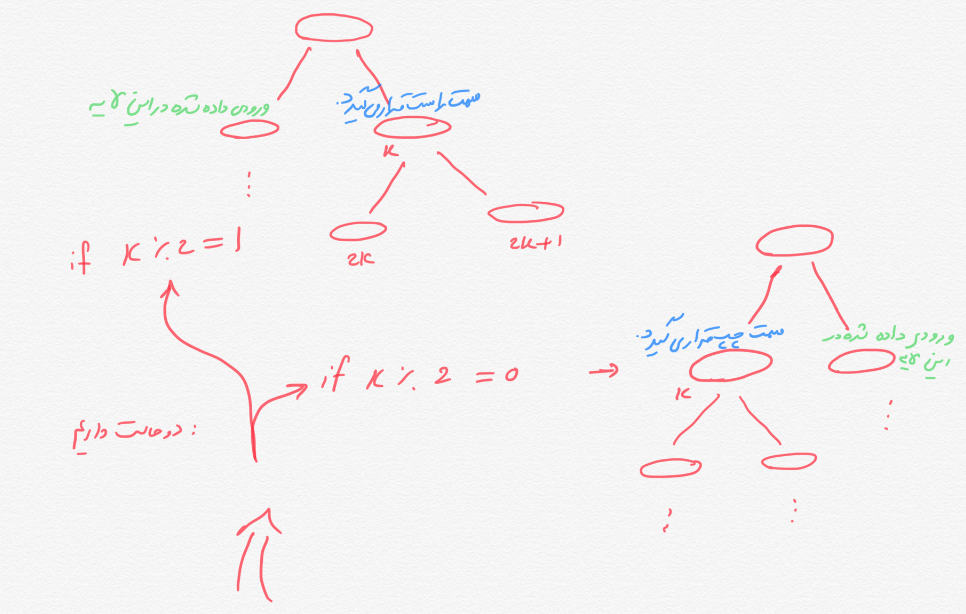
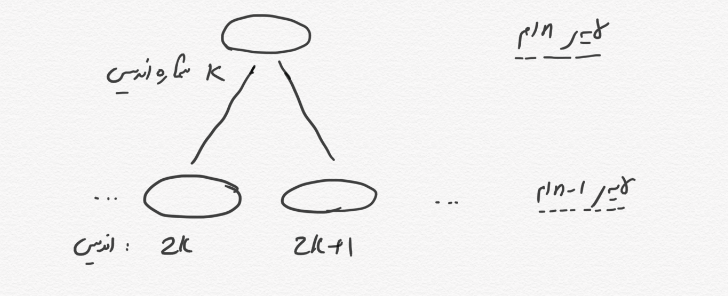
توجه:طبق اصلاحیه مطرحی برای سوال 3،در ابتدا اندیسِ فایلی که قرار است صحت اش مورد ارزیابی قرار بگیرد را وارد می کنیم،مثلا در صورت سوال،جهتِ ارزیابی فایلی که کادر آبی دور آن کشیده شده است،لازم است تا ابتدا اندیسِ 4 وارد شود(اندیس ها از 0 شروع می شوند)و سپس در ادامه باقیِ فایل هارا با ترتیبی مشخص،ارسال می کند.

نکته کلیدی در حل این قسمت،نحوه concat کردنِ مقدار hash ِنود های لایه پایین جهت محاسبه مقدار نودِ لایه بالاتر است(یعنی هنگامِ concat کردن،چپ و راست نشوند)،برای تشخیص اینکه مقدار نود فعلی،سمت راست ورودی داده شده در این لایه قرار بگیرد یا سمت چپ آن از اندیس آن در لایه استفاده می کنیم.

نکته 1:فرض کنید که نود با اندیس j در لایه k قرار دارد،داریم که شماره اندیس پدر این نود در لایه k-1 برابر است با:

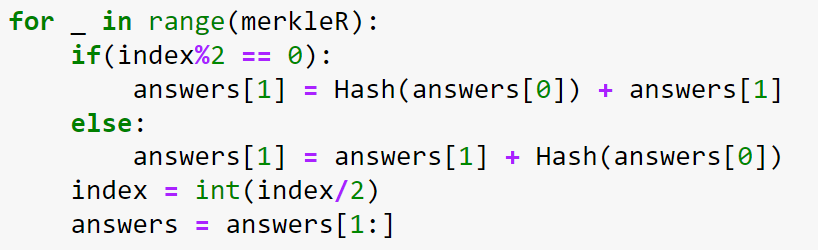
علامت [] به معنیِ جزصحیح است.

حال به کمک نکته ذکر شده مساله را حل می کنیم،به نوشته زیر توجه کنید:



حال قطعه کد زیر،پیاده سازی شده یِ نوشته بالاست:

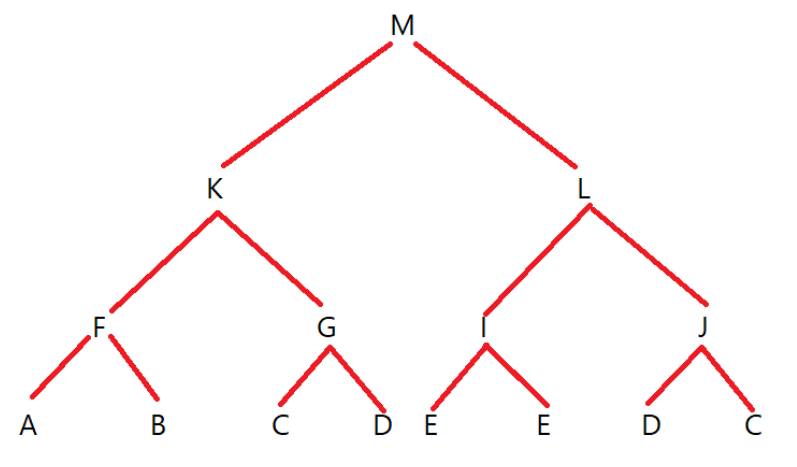
آرایه answers همان فایل های داده شده در ورودی merkle tree proof است.



همانطور که مشاهده می کنید به ترتیب لایه هارا بالا میرویم تا سرانجام مقدار root ِ درخت merkle محاسبه گردد و سپس با مقدارِ merkle root داده شده مقایسه می کنیم و در صورت برابری “Existed” می دهیم و در غیراینصورت “Did not exist” می دهیم.

مثال۱:

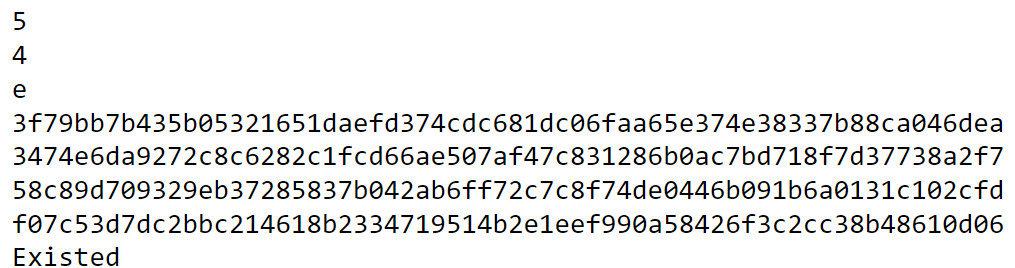
درخت زیر را درنظربگیرید که به دنبال احراز اصالت e با اندیس 4 هستیم:



ورودی ها به شرح زیر است:

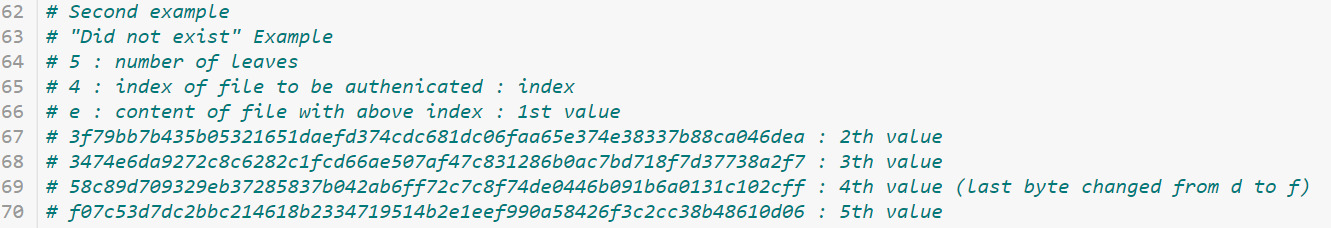


حال مقادیر فوق را به عنوان ورودی به کد می دهیم و خروجی اش را مشاهده می کنیم:

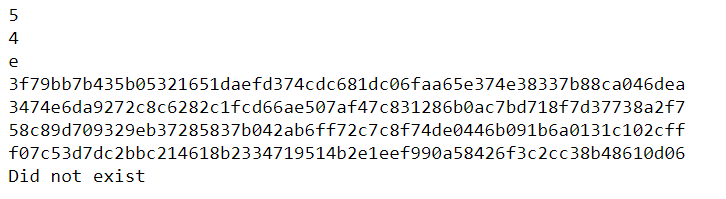


همانطور که انتظار می رفت فایل وجود دارد.توجه درخت این مثال همان درخت مثال سوال ۲ است و همانطور که مشاهده می کنید مقدار merkle root محاسبه شده برابر همان merkle root سوال ۲ شده است و لذا طبق انتظار فایل وجود دارد.

مثال۲:



دقت کنید که ما بایت آخر 4th value را از d به f تغییر دادیم لذا انتظار می رود که خروجی “Not Existed” باشد،مقادیر را به برنامه می دهیم و خروجی اش را ملاحظه می کنیم:



و خروجی همانطور است که انتظارش را داشتیم.