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.

```python
# A
def H(number):
    string = str(number)
    return hashlib.sha256(string.encode()).hexdigest()[-5:]

myId = 96101635
collision = myId + 1
while(True):
    if(H(collision) == H(myId)):
        print(collision)
        break
    else:
        collision = collision + 1
```

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.

```python
print("H(96101635) = " + str(H(96101635)))
print("H(98712725) = " + str(H(98712725)))
```

```
H(96101635) = 2a4b6
H(98712725) = 2a4b6
```

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 $D = 2^{20}$).

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:

$$\left(\frac{D-1}{D}\right)^{k-1}$$

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:

$$1-\left(\frac{D-1}{D}\right)^{k-1}$$

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:

$$E[number\ of\ matches\ in\ n] = \sum_{k=1\ to\ n} \{1-\left(\frac{D-1}{D}\right)^{k-1}\}$$

$$E[number\ of\ matches\ in\ n] = n - \sum_{k=1\ to\ n} \{(\frac{D-1}{D})^{k-1}\}$$

ادامه در صفحه بعد.

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

$$\sum_{k=0}^{m} r^k = \frac{1 - r^{m+1}}{1 - r}$$

حال داریم که:

$$\sum_{k=1}^{n} \left(\frac{D-1}{D}\right)^{k-1}$$

$$= \sum_{k=0}^{n-1} \left(\frac{D-1}{D}\right)^{k}$$

$$= \frac{1 - \left(\frac{D-1}{D}\right)^{n}}{1 - \frac{D-1}{D}}$$

$$= \frac{1 - \left(\frac{D-1}{D}\right)^{n}}{\frac{D}{D} - \frac{D-1}{D}}$$

$$= \frac{1 - \left(\frac{D-1}{D}\right)^{n}}{\frac{D-D+1}{D}}$$

$$= \frac{1 - \left(\frac{D-1}{D}\right)^{n}}{\frac{1}{D}}$$

$$= D\left(1 - \left(\frac{D-1}{D}\right)^{n}\right)$$

$$= D - D\left(\frac{D-1}{D}\right)^{n}$$

ادامه در صفحه بعد.

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

$$E[number\ of\ collisions] = n - D + D\left(\frac{D-1}{D}\right)^n$$

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)

```
# this numbers have been extracted from different runs of the above code.(part B's code)
print("H(763181) = " + str(H(763181)))
print("H(743982) = " + str(H(743982)))

print("H(631908) = " + str(H(631908)))
print("H(998692) = " + str(H(998692)))

print("H(512732) = " + str(H(512732)))
print("H(104557) = " + str(H(104557)))

print("H(703967) = " + str(H(703967)))
print("H(541384) = " + str(H(541384)))
```

```
H(763181) = 138a9
H(743982) = 138a9
H(631908) = 77fbf
H(998692) = 77fbf
H(512732) = 05433
H(104557) = 05433
H(703967) = 9a174
H(541384) = 9a174
```

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:

a. 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.

b. 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:

```
6  def sender(message,secretKey):
7      messageHash = Hash(message + secretKey)
8      output = [message,messageHash]
9      return output
10
11 def receiver(sentMessage,secretKey):
12     mainM = sentMessage[0]
13     msHash = sentMessage[1]
14     if(Hash(mainM + secretKey) == msHash):
15         return mainM
16     else:
17         return "The message has been changed!"
```

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:

```python
# example everything is fine
s = "SimpleKey"
m = "Hi everybody,Are you all ok?"
sentM = send(m,s)
result = receiver(sentM,s)
print(result)

# example2: using different secret_key(secretKey(sender)!=secretKey(receiver))
s = "SimpleKey"
m = "Hi everybody,Are you all ok?"
sentM = send(m,s)
result = receiver(sentM,s + "#")
print(result)

# example3: mainMessage changed
s = "SimpleKey"
m = "Hi everybody,Are you all ok?"
sentM = send(m,s)
sentM[0] = "changed Message"
result = receiver(sentM,s)
print(result)
```

```
Hi everybody,Are you all ok?
The message has been changed!
The message has been changed!
```

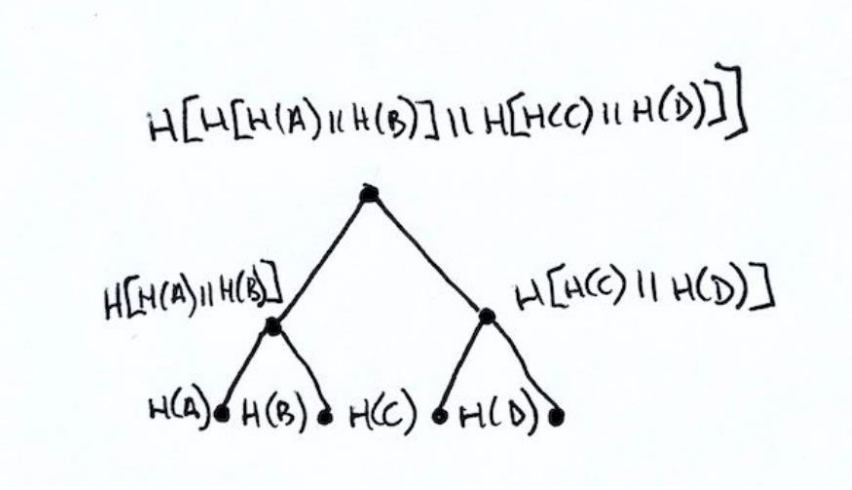
سوال دوم:

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

```python
import math
import hashlib

n = int(input().strip())
merkleR = int(math.log2(n))
if(not(2**merkleR == n)):
    merkleR = merkleR + 1
messages = []

for i in range(n):
    messages.append(input().strip())

def upgradeLeaves(ms):
    temp = []
    for i in range(0,2**merkleR - len(ms)):
        temp.append(ms[-(i+1)])
    ms.extend(temp)
    print(ms)

```

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).



درکد 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)

```python
def addLayerHashes(layer,ms):
    if(layer == 0):
        return [Hash(ms[0]) + Hash(ms[1])]
    else:
        return [Hash(ms[2*i]) + Hash(ms[2*i+1]) for i in range(2*layer)]
```

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.

```
29 def calculateRoot(ms):
30     depth = int(math.log2(len(ms)))
31     for layer in range(depth):
32         ms = addLayerHashes((depth-1)-layer,ms)
33     print("Hex output of root: ")
34     return Hash(ms[0])
35     print("Hex output of root: ")
36     print(ms)
```

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 rootِ درخت merkle را در خروجی می نویسیم:

```
5
a
b
c
d
e
Merkle tree Leaves:
['a', 'b', 'c', 'd', 'e', 'e', 'd', 'c']
Hex output of root:

'f07c53d7dc2bbc214618b2334719514b2e1eef990a58426f3c2cc38b48610d06'
```

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.

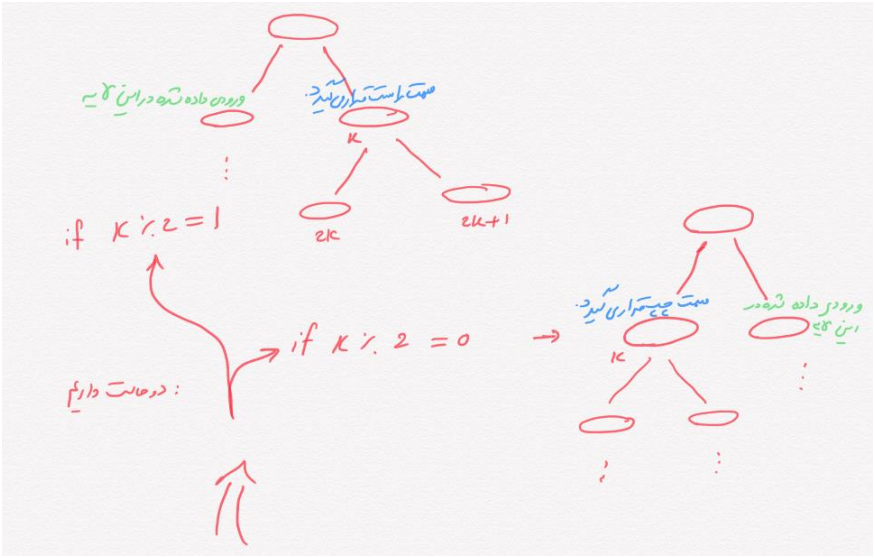
نکته کلیدی در حل این قسمت،نحوه concat کردنِ مقدار hash ِنود های لایه پایین جهت محاسبه مقدار نودِ لایه بالاتر است(یعنی هنگامِ concat کردن،چپ و راست نشوند)،برای تشخیص اینکه مقدار نود فعلی،سمت راست ورودی داده شده در این لایه قرار بگیرد یا سمت چپ آن از اندیس آن در لایه استفاده می کنیم.

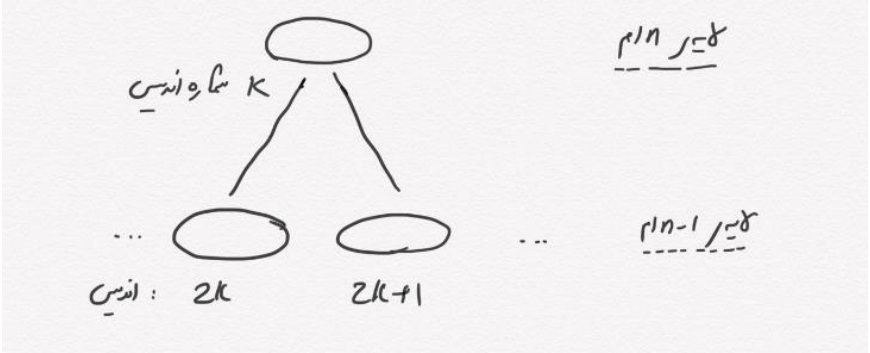
نکته 1:فرض کنید که نود با اندیس j در لایه k قرار دارد،داریم که شماره اندیس پدر این نود در لایه k-1 برابر است با:

$$index\ of\ father = \left[\frac{j}{2}\right]$$

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:

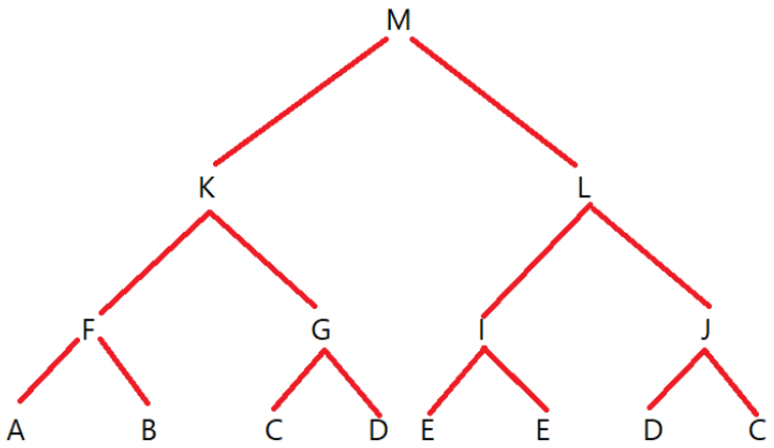
آرایه answers همان فایل های داده شده در ورودی merkle tree proof است.

```
for _ in range(merkleR):
    if(index%2 == 0):
        answers[1] = Hash(answers[0]) + answers[1]
    else:
        answers[1] = answers[1] + Hash(answers[0])
    index = int(index/2)
    answers = answers[1:]
```

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:

```
# "exist" Example
# 5 : number of leaves
# 4 : index of file to be authenicated : index
# e : content of file with above index : 1st value
# 3f79bb7b435b05321651daefd374cdc681dc06faa65e374e38337b88ca046dea : 2th value
# 3474e6da9272c8c6282c1fcd66ae507af47c831286b0ac7bd718f7d37738a2f7 : 3th value
# 58c89d709329eb37285837b042ab6ff72c7c8f74de0446b091b6a0131c102cfd : 4th value
# f07c53d7dc2bbc214618b2334719514b2e1eef990a58426f3c2cc38b48610d06 : 5th value
```

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

```
5
4
e
3f79bb7b435b05321651daefd374cdc681dc06faa65e374e38337b88ca046dea
3474e6da9272c8c6282c1fcd66ae507af47c831286b0ac7bd718f7d37738a2f7
58c89d709329eb37285837b042ab6ff72c7c8f74de0446b091b6a0131c102cfd
f07c53d7dc2bbc214618b2334719514b2e1eef990a58426f3c2cc38b48610d06
Existed
```

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:

```
# Second example
# "Did not exist" Example
# 5 : number of leaves
# 4 : index of file to be authenicated : index
# e : content of file with above index : 1st value
# 3f79bb7b435b05321651daefd374cdc681dc06faa65e374e38337b88ca046dea : 2th value
# 3474e6da9272c8c6282c1fcd66ae507af47c831286b0ac7bd718f7d37738a2f7 : 3th value
# 58c89d709329eb37285837b042ab6ff72c7c8f74de0446b091b6a0131c102cff : 4th value (last byte changed from d to f)
# f07c53d7dc2bbc214618b2334719514b2e1eef990a58426f3c2cc38b48610d06 : 5th value
```

دقت کنید که ما بایت آخر 4^{th} 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:

```
5
4
e
3f79bb7b435b05321651daefd374cdc681dc06faa65e374e38337b88ca046dea
3474e6da9272c8c6282c1fcd66ae507af47c831286b0ac7bd718f7d37738a2f7
58c89d709329eb37285837b042ab6ff72c7c8f74de0446b091b6a0131c102cff
f07c53d7dc2bbc214618b2334719514b2e1eef990a58426f3c2cc38b48610d06
Did not exist
```

و خروجی همانطور است که انتظارش را داشتیم.