O mundo descentralizado

Começando com a Arpanet, filha direta da guerra fria, (suprema ironia: como a humanidade progride quando precisa matar mais a custos menores...) o mundo caminhou nos últimos 50 anos em direção à descentralização. Aqui a lei fundamental é que não há um lugar privilegiado, todos são iguais, e sobretudo, todos são descartáveis. Repare na disponibilidade desta arquitetura e consequentemente na sua confiabilidade. É tudo o que a gente precisa.

Uma belíssima construção teórico/prática é a do Bitcoin: é uma arquitetura admirável, trouxe inúmeras contribuições brilhantes, e deu origem a dezenas de tecnologias poderosas ainda mais quando usadas em conjunto. Glória e exaltação a Satoshi Nakamoto, seja ele quem for. Como diz a wikipedia:

> Estima-se que Nakamoto tenha uma fortuna de aproximadamente um milhão de bitcoins. A verdadeira identidade de Nakamoto permanece desconhecida, e tem sido objeto de muita especulação. Não se sabe se o nome "Satoshi Nakamoto" é real ou um pseudônimo, ou se o nome de uma pessoa ou representa um grupo de pessoas.

Talvez a principal constribuição seja a do Blockchain, um conjunto de certificações descentralizadas e rastreáveis. À medida em que as transações vão sendo efetuadas elas são registradas em um bloco por milhares de gestores/mineradores. Quando o bloco for encerrado, mineradores entram em mineração deste bloco e quando o primeiro minerador for bem sucedido, o bloco é disseminado e incluído na blockchain por todos os participantes.

Este exercício vai se concentrar numa tecnologia básica do Blockchain que é a certificação: como garantir que os (pedaços de) blocos trocados na rede sejam todos válidos e não resultado de algum pilantra tentando iludir a rede e ganhar algum. Como os blocos tendem a crescer em tamanho, necessita-se um algoritmo de certificação eficiente e que cresça menos em exigência computacional do que o tamanho dos blocos. Quem faz isso é o algoritmo da árvore de Merkle. Ao invés de crescer em proporção ao tamanho n do bloco, ele cresce em proporção a $log_2\ n$, o que é muito melhor. Acompanhe: O $log_2\ de\ 10 = 3.3$, o $log_2\ de\ 100 = 6.6$, e o $log_2\ de\ 1000000 = 19.9$.

Árvores de Merkle

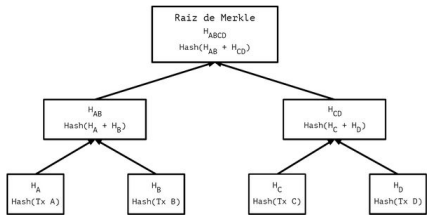
Inventadas em 1979 por Ralph Merkle, e devidamente patenteadas (patente US4309569 A) é uma estrutura de dados que serve para garantir a integridade de um bloco maior de dados. É uma extensão do conceito de Hash.

O hash é uma assinatura binária, obtida a partir de aplicação de um algoritmo a um objeto binário. É um cálculo simples (?!) em uma direção, do blob → hash, mas impossível na direção oposta: hash → blob.

Neste contexto, as árvores de Merkle são usadas para garantir que blocos de dados recebidos de outros pares, em uma rede P2P, são recebidos intactos e inalterados. É usada para verificar se outros pares não mentem e enviam blocos falsos.

A principal diferença do uso simples de um hash para verificar a integridade de um bloco de dados é que cada ramo (pedaço) pode ser verificado imediatamente, embora toda a árvore não esteja disponível. Esta separação é eficiente pois permite que apenas pequenos blocos sejam baixados novamente caso estejam danificados.

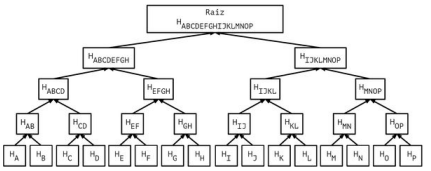
No bloco do bitcoin (que lembrando é formado pelas transações aceitas no período de tempo do bloco) vai a raiz da árvore de Merkle. Todas as transações são dispostas em uma árvore binária. Cada transação tem o seu id que é o hash da transação concatenado com id da transação irmã. veja-se numa figura



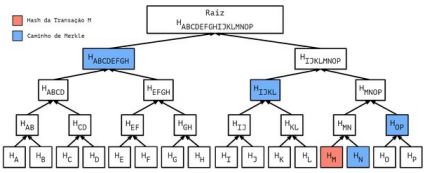
Para isso, todas as transações devem ser colocadas no mesmo nível como na imagem e formarem duplas para criarem os níveis acima sucessivamente. Em caso do bloco não ter um número par de transações, tudo que se faz é repetir a última transação.

Obviamente, no Bitcoin, estas árvores são muito maiores e proporcionais ao número de transações de cada bloco e em cenários desta magnitude que esta estrutura se torna uma solução para comprovação eficiente da existência e integridade de uma transação num dado bloco. Cada hash deste tem o tamanho de 32 bytes e a complexidade de busca na árvore de Merkle cresce $O(log_2(N))$ na notação "Big-O"com N sendo o número de transações.

Vejamos um exemplo um pouco maior de uma árvore de Merkle criada a partir de 16 transações:



Caso precisemos comprovar que a transação M está incluida no bloco, precisamos de apenas 4 hashes de 32 bytes num total de 128 bytes para formar o nosso caminho de Merkle. Veja como o caminho de Merkle junto com a raiz de Merkle é tudo que precisamos para comprovar que um bloco inclui a transação M:



A eficiência da árvore de Merkle para este objetivo vai se tornando mais óbvia à medida em que aumenta-se o número de transações e comparamos com o número de bytes necessários para comprovar a existência de uma transação nestes números maiores:

Transações	Tamanho Aproximado do Bloco	Caminho de Merkle (hashes)	Caminho de Merkle (bytes)
16	4 K	4	128
512	128 K	9	288
2048	512 K	11	352
65.525	16 M	16	512

O que se observa é que com poucas transações não parece fazer muito sentido o uso da árvore de Merkle, mas logo que o número de transações começa a saltar podemos ver claramente a otimização que esta estrutura traz ao sistema do Bitcoin.

☞ Para você fazer

Vamos simular uma Árvore de Merkle usando versos de Os Lusíadas, ao invés de blocos de transações. A idéia entretanto, permanece a mesma. Considere primeiro definir uma função simples de *hashing* usando 5 bytes de resultado e usando também o sistema de numeração de base 58, formado pelos seguintes valores: (Porquê 58?: 26+26+10=62, menos zero, I, O e l).

```
          1         2         3
12345678901234567890123456789012345678901
123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdef
4         5
01234567890123456789
ghijkmnopqrstuvwxyz
```

Considere a seguinte função de numeração em base 58:

```
def n58(x):
  print('-----',x)
  b58="123456789ABCDEFGHJKLMNPQRST
    UVWXYZabcdefghijkmnopqrstuvwxyz"
  a1=x//(58**4)
  c5=b58[a1]
  a2=(x-((58**4)*a1))//(58**3)
  c4=b58[a2]
  a3=(x-(((58**4)*a1)+((58**3)*a2)))//(58**2)
  c3=b58[a3]
  a4=(x-(((58**4)*a1)+((58**3)*a2)+((58**2)*a3)))
    //58
  c2=b58[a4]
  a5=x-((((58**4)*a1)+((58**3)*a2)+((58**2)*a3))
    +58*a4)
  c1=b58[a5]
  return c5+c4+c3+c2+c1
```

Para testar sua função veja lá:
n58(ASARMASEOSBAROESASSINALADOS) é Ty7jc
n58(QUEDAOCIDENTALPRAIALUSITANA) é yjCgo
n58(PORMARESNUNCADEANTESNAVEGADOS) é EJAJZ

Eis aqui a função que calcula o hash e depois converte-o em um número de base 58:

```
def hs(x):
  ta=len(x)
-------------------------------------------------
  i=0
  a=656356701
-------------------------------------------------
    z=ord(x[i])*i
    k=k+z+a
    k=k%656356767 # =(58^5)-1
-------------------------------------------------
    i=i+1
  return n58(k)
```

1. Agora, imagine que você quer saber se a frase VOSPODEROSOREICUJOALTOIMPERIO faz parte ou não do bloco de controle. Para sua informação ela deve estar na árvore no nodo: 53 e os hash dos nodos irmão que interessam são 52=FrRHt, 27=11FLi, 12=11G9w, 7=11BMc, 2=11EPf, 1=11D3i

2. Frase SESERDOMUNDOREISEDETILGENTE faz parte ou não do bloco de controle ? Para sua informação ela deve estar na árvore no nodo: 62 e os hash dos nodos irmão que interessam são 63=TAtDb, 30=11FQK, 14=11FW3, 6=11Cdv, 2=11EPf, 1=11D3i

3. Frase ASARMASEOSBAROESASSINALADOS faz parte ou não do bloco de controle? Para sua informação ela deve estar na árvore no nodo: 36 e os hash dos nodos irmão que interessam são 37=PAwWP, 19=11Eza, 8=11EdX, 5=11DV6, 3=11EYi, 1=11D3i

4. Frase QUEOUTROVALORMAISALTOSEALEVANTA faz parte ou não do bloco de controle? Para sua informação ela deve estar na árvore no nodo: 44 e os hash dos nodos irmão que interessam são 45=Hk9fH, 23=11Mte, 10=11By9, 4=11Dkg, 3=11EYi, 1=11D3i

Responda aqui (não: responda 0 e sim: responda 1)

1	2	3	4