Aula TP - 25/Mar/2019

Exercícios

1. Blockchain

Pergunta 1.1

O código do ficheiro "main.experiencia1.1.js" foi alterado de modo a que o primeiro bloco do método que cria o Genesis Block tivesse como *timestamp* a data em que o mesmo foi criado sendo, assim, o código seguinte a maneira de fazer o mesmo:

```
createGenesisBlock(){
        var ts = new Date();
        return new Block(0, ts.toDateString(), "Bloco inicial da koreCoin", "0");
    }
```

Sendo que o *timestamp* fica da seguinte maneira `Fri Apr 12 2019`.

Pergunta 1.2

```
koreCoin.addBlock(new Block (1, "01/01/2018", {amount: 20}));
koreCoin.addBlock(new Block (2, "02/01/2018", {amount: 40}));
koreCoin.addBlock(new Block (3, "02/01/2018", {amount: 60}));
koreCoin.addBlock(new Block (4, "03/01/2018", {amount: 80}));
koreCoin.addBlock(new Block (5, "03/01/2018", {amount: 100, Bloco1: 40, Bloco2: 20}));
```

2. Proof of Work Consensus Model

Pergunta 2.1

Dificuldade 2:

```
Mining block 1...

Iterations needed to mine: 199
Block mined: 00efedbe59cd9d01c80eefdf6d191b354bfdc38834cdd17568e2d3148415c41f
Mining block 2...

Iterations needed to mine: 80
Block mined: 0032ef45c1552221e49a3c3927a8e5721d11c224f75908057bbf1518c11eb226
Mining block 3...

Iterations needed to mine: 689
Block mined: 009e3a69807138a9bd373c4667bbfd6f6881042a9445cae5581b9f4c5f8b2206
Finished!
node main.js  0,18s user 0,02s system 54% cpu 0,361 total
```

Dificuldade 3:

```
Mining block 1...

Iterations needed to mine: 624
Block mined: 0009cc6230e93dca2938b6a010038687641c2faad82a8255ee54cc6fb959883e
Mining block 2...

Iterations needed to mine: 2202
Block mined: 000ea01b16b05582765114b9ef44a0543a7e58cfe02a1a83205bb7806626becf
Mining block 3...

Iterations needed to mine: 4130
Block mined: 00027b89dec7c85ca50958a236a249b09a0563f72e1e0059ba4b78f1aa4f3c8d
Finished!
node main.js  0,24s user 0,04s system 113% cpu 0,249 total
```

Dificuldade 4:

```
Mining block 1...

Iterations needed to mine: 16762
Block mined: 0000330a027a3e3972883e830bcbf0f941e725625868febae45e57247e7b2bbf
Mining block 2...

Iterations needed to mine: 115348
Block mined: 00006cfc259b1bcdacd82f9acdc8613619c708cd7f37c787107def1c292f94df
Mining block 3...

Iterations needed to mine: 19253
Block mined: 0000fe0fe98d84f68f555fe88bd93c8bf9b1d3304b0a962efbecade308bdd7bf
Finished!
node main.js  1,95s user 0,06s system 107% cpu 1,880 total
```

Dificuldade 5:

```
Mining block 1...
Iterations so far: 1000000

Iterations needed to mine: 1738828
Block mined: 00000167609f6054daf10c79502a51b13b06b0c3c2fc3b7b08c8de82c23beba6
Mining block 2...

Iterations needed to mine: 747260
Block mined: 00000939dc98d09bf4636fb7281dee3a2af59b70087734344d2648c6e97eff8d
Mining block 3...

Iterations needed to mine: 110214
Block mined: 00000e5bb489bbf7a29d3153eb71e6d012023d995ca6def7f47d387ac9c1cd1e
```

```
Finished!
node main.js  30,82s user 0,52s system 103% cpu 30,233 total
```

Conclusão:

O tempo de encontrar um Hash que contenha exatamente um determinado número de zeros no início (que é a dificuldade determinada) aumenta de forma exponencial conforme a dificuldade aumenta. Interessa notar que uma criptomoeda que trabalhe com um Proof-of-Work dessa forma permite que os primeiros *miners* consigam minerar os primeiros blocos de forma relativamente fácil. Além disso, é possível perceber que é inviável para um computador comum fazer a mineração de blocos de criptomoedas já estabelecidas, como é o caso do Bitcoin.

Pergunta 2.2

1

O algoritmo *proof of work* utilizado é simples: para criar um novo bloco cada *miner* tem que incrementar um número a um número de prova anterior e quando esse número for divisível por 9 e divisível pelo número da prova do anterior bloco, um novo bloco será criado, e o *miner* terá a sua recompensa.

2

Para que um algoritmo de mineração seja adequado, o *miner* tem que efetuar o puzzle computacionalmente intensivo e ser recompensado por esse trabalho, normalmente na criptomoeda nativa que faz parte do sistema de consenso. Caso contrário, não haveria confiança na criptomoeada dado que utilizadores maliciosos poderiam tentar manipular a criptomoeda.

Esse puzzle tem que ser difícil de resolver, mas fácil de verificar que a solução é válida, como também todos os nodos conseguem validar quaisquer próximos blocos propostos e, qualquer próximo bloco que não satisfaça o puzzle pode ser rejeitado.

Devido à probabilidade muito baixa de geração bem-sucedida, isso torna imprevisível qual *miner* na rede será capaz de gerar o próximo bloco, uma vez que necessita de processos aleatórios com baixa probabilidade e requer muita tentativa-e-erro sem qualquer média antes de se gerar a prova. Por isso, o trabalho envolvido num puzzle não influencia a probalilidade de resolver o puzzle atual ou futuro, i.e., os puzzles são independentes. A utilização de *nonces* no *hash* introduz essa inprevisibilidade.

Para que um bloco seja válido, ele deve ter um valor menor que o destino atual. Isso significa que cada bloco indica o trabalho feito gerando-o. Cada bloco contém o *hash* do bloco anterior. Assim, cada bloco tem uma cadeia de blocos que contém uma quantidade significativa de trabalho.

Após algumas iterações dá para perceber o padrão descrito na pergunta anterior: todos os números são múltiplos de 9 e a prova anterior divide a prova corrente, sendo que a prova do próximo bloco será o mais próximo múltiplo de 9. De facto a tabela abaixo dá para mostrar matemáticamente o cálculo de todas as provas, dependendo do índice do bloco. Seja um $k \geq 0$ o índice atual, a prova corrente será $2^k \times 3^2 \equiv 2^k \times 9$. Este resultado pode ser comprovado para um $k \in [0, \ldots, 26]$ na tabela seguinte.

Este resultado pode ser obtido pela fatorização do número da prova e pela análise do algoritmo para os primeiros blocos gerados. Embora a fatorização seja um problema difícil, os números envolvidos são muito pequenos e, até que a *blockchain* atinja um número de blocos em que o número seja difícil de fatorizar pode levar bastante tempo, como também, deve-se assumir que o algoritmo é do conhecimento do atacante.

index	proof-of-work	Cálculo
0	9	$2^0 \times 3^2$
1	18	$2^1 \times 3^2$
2	36	$2^2 \times 3^2$
3	72	$2^3 \times 3^2$
4	144	$2^4 \times 3^2$
5	288	$2^5 \times 3^2$
6	576	$2^6 \times 3^2$
7	1152	$2^7 \times 3^2$
8	2304	$2^8 \times 3^2$
9	4608	$2^9 \times 3^2$
10	9216	$2^{10} \times 3^2$
11	18432	$2^{11} \times 3^2$
12	36864	$2^{12} \times 3^2$
13	73728	$2^{13} \times 3^2$
14	147456	$2^{14} \times 3^2$
15	294912	$2^{15} \times 3^2$
16	589824	$2^{16} \times 3^2$
17	1179648	$2^{17} \times 3^2$
18	2359296	$2^{18} \times 3^2$
19	4718592	$2^{19} \times 3^2$
20	9437184	$2^{20} \times 3^2$
21	18874368	$2^{21} \times 3^2$
22	37748736	$2^{22} \times 3^2$
23	75497472	$2^{23} \times 3^2$
24	150994944	$2^{24} \times 3^2$
25	301989888	$2^{25} \times 3^2$
26	603979776	$2^{26} \times 3^2$

Ou seja, a razão para o algoritmo não ser válido é a previsibilidade do trabalho futuro e o atual, ou seja o puzzle não é independente porque podemos prever o trabalho a partir do índice.