Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Campus Ponta Grossa

Projeto Final de Estruturas de Dados 1 e 2:
Simulação de BlockChain Simplificada

Prof. Saulo Queiroz

1 Introdução

Este documento explanar os conceitos básicos, o enunciado e os resultados esperados para o projeto final da disciplina de estruturas de dados. O projeto consiste na simulação de uma versão simplificada da blockchain da criptomoeda bitcoin (BTC).
Nota: algumas aulas serão dedicadas para auxiliar o desenvolvimento do projeto
Em caso de dúvidas ou erros no enunciado, favor contactar o professor.

2 Informações Gerais: Entrega e Formação de Grupos

2.1 Formação dos Grupos

Quantidade de Grupos: 8, identificados por um código de 0 a 7
Total de integrantes por grupo: 4
Fórmula para determinar código do seu grupo: (RA+tentativa) mod 8, onde, RA é seu registro acadêmico, mod é a função resto da divisão inteira, tentativa começa com zero. Caso você seja direcionado para um grupo que já tenha 4 integrantes e tenha o maior RA do grupo, você deverá procurar um novo grupo. Para isso, incremente a variável 'tentativa' e recalcule a fórmula do código de grupo. Repita o processo até que você entre num grupo com apenas 4 RAs ou seu RA esteja entre os 4 menores de grupo. Em caso de dúvidas, contacte o professor.

2.2 O Que e Quanto Entregar

Data da entrega: Conferir Moodle
.

Arquivo .zip contendo:
– código fonte em C do simulador da blockchain do bitcoin
– arquivo de texto com todos os dados de todos os blocos minerados.
Alunos de ED2 devem também entregar o arquivo binário contendo a blockchain.
– seu código deverá apresentar um menu que fornece as respostas para os seguintes dados da sua blockchain:
a) endereço com mais bitcoins e o número de bitcoins dele (liste mais de um em caso de empate)
b) endereço que minerou mais blocos (liste mais de um em caso de empate)
c) hash do bloco com mais transações e o número de transações dele (liste mais de um em caso de empate)
d) hash do bloco com menos transações e o número de transações dele (liste mais de um em caso de empate)

e) quantidade média de bitcoins por bloco

As respostas dos itens acima deverão ser dadas também em um PDF a ser enviado dentro do .zip.

– inclua também no seu menu opções que executam as seguintes funções
f) imprimir todos os campos de um bloco dado o número do bloco. Para alunos de ED2, deverá ser feita uma pesquisa eficiente em arquivo binário considerando que os blocos minerados estão sequencialmente armazenados;
g) imprimir todos os campos dos n primeiros blocos minerados por um dado endereço. Ou seja, esta consulta requer dois dados de entrada (n e o endereço do minerador). Observação para alunos de ED2: Esta consulta deve ser feita a partir de um arquivo de índices construído sobre o campo endereço. Se o arquivo não existir na primeira vez que a opção for escolhida, construa-o e carregue-o em memória RAM para acelerar as consultas. Preze pela eficiência na escolha de sua ED em RAM.
h) imprimir todos os campos dos n primeiros blocos (n é dado de entrada). A impressão deve ser em ordem crescente de quantidade de transações do bloco.
i) imprimir todos os campos de todos os blocos que têm um dado nonce. Observação para alunos de ED2: Esta consulta deve ser feita a partir de um arquivo de índices construído sobre o campo nonce. Se o arquivo não existir na primeira vez que a opção for escolhida, construa-o e carregue-o em memória RAM para acelerar as consultas. Preze pela eficiência na escolha de sua ED em RAM.

3 Conceitos Básicos e Como Implementá-los

A seguir, explanamos os seguintes conceitos necessários ao projeto.

- **código (criptográfico) hash**: valor númerico (geralmente expresso em hexadecimal) que representa de forma única uma determinada informação. Como no bitcoin, neste projeto deverá ser usado o algoritmo SHA-256. Na implementação da biblioteca Open SSL que deverá ser adotada para este projeto, um código hash SHA-256 cabe em um vetor `unsigned char` de 32 posições (confira o exemplo provido pelo professor);

- **endereço**: número que identifica um usuário do sistema financeiro da blockchain (é uma espécie de CPF do sistema). Para fins de simplificação, consideramos apenas 256 usários representados por endereços de `0` a `255`. Ou seja, o endereço deve obrigatoriamente ser do tipo `unsigned char`;

- **transação**: um evento qualquer em que um endereço (chamado end. de origem) envia a um end. de destino uma determinada quantidade de bitcoins. Para simplificar, considera-se que a quantidade de bitcoins é um inteiro curto de 0 a 50, ou seja, deve obrigatoriamente ser do tipo `unsigned char`. No exemplo a seguir, temos uma transação em que o endereço `0` envia `50` bitcoins ao endereço `255`: `0 255 50`. Quando essa transação for validada (explicação adiante), os saldos dos endereços envolvidos na transação devem ser atualizados.

- **carteira do sistema**: mantem o saldo da quantidade de bitcoins de todos os 256 endereços do sistema. Deve ser implementado com o vetor `unsigned int carteira[256]` (inicializado com zeros) em que a posição `i` guarda a quantidade de bitcoins do endereço `i`. Ou seja, o índice do vetor é o endereço. Todas as posições devem ser inicializadas com zero. Ex.: Após a transação `15 7 10` ser validada no sistema, teremos `carteira[15]-=10; carteira[7]+=10;`. **Somente endereços que têm bitcoins podem ser endereços de origem. Para um endereço ter bitcoins, ele deve ter sido recom-**

pensado por minerar um bloco ou ter recebido bitcoins de alguém que minerou um bloco (explicaremos a mineração adiante);

- **Bloco Não Minerado**: Registra um conjunto finito de transações que representam os negócios dos usuários em um dado momento. Deve ser composto dos seguintes campos:
  - `numero`: número sequencial do bloco. Começa em 1;
  - `nonce`: Deve variar de 0 até 4294967295 ($2^{32} - 1$) para permitir a validação das transações do bloco (conforme explicaremos depois);
  - `data[184]`: Vetor que conterá as transações do bloco e o endereço do minerador (que deverá ser escolhido aleatoriamente do intervalo [0,255] e sempre armazenado na última posição do vetor data, mais detalhes na seção 4). Se um bloco tiver duas transações (Ex. `1 2 3` e `4 5 6`) e for minerado pelo minerador `125`, o vetor data deve ser assim preenchido: `data[0]=1; data[2]=2; data[2]=3; data[3]=4; data[4]=5; data[5]=6; data[183]=125.` Todas as demais posições devem ser zeradas. Após a mineração, o minerador deverá ser recompensado com 50 bitcoins na carteira do sistema, ou seja, neste exemplo teríamos `carteira[125]+=50;`.

    A quantidade de transações e os valores dela serão determinados aleatoriamente (explicação na seção 4.1)
  - `hashAnterior[SHA256_DIGEST_LENGTH]`: Valor do código hash do bloco anterior já minerado (explicação a seguir). O valor `SHA256_DIGEST_LENGTH` é uma constante de valor igual a 32 (bytes) definida pela biblioteca Open SSL.
- **Mineração, Minerador e Recompensa**: As transações de um bloco não podem ser refletidas na carteira do sistema até que o bloco que as contêm seja minerado, isto é, validado. Um bloco é válido se seu código o código hash SHA-256 começar com dois zeros hexadecimais (no Bitcoin essa quantidade é dinâmica). Ou seja, se o código hash SHA-256 de um bloco estiver no vetor `hash` de 32 posições, então o bloco será considerado válido se `hash[0]==0`. Caso contrário, o campo `nonce` deverá ser incrementado e um novo hash calculado até que o bloco seja minerado com sucesso.
- **Blockchain**: É uma sequência de blocos já minerados. Note que o campo-vetor `hashAnterior` do bloco número i deverá conter o hash que validou o bloco número $i - 1$. No caso do bloco número 1, esse campo `hashAnterior` deve ser inteiramente preenchido com zeros. A struct de um bloco minerado é composta de uma variável do tipo `BlocoNaoMinerado` mais o hash que o valida, ou seja,

```
typedef struct BlocoMinerado
{
  BlocoNaoMinerado bloco;
  unsigned char hash[SHA256_DIGEST_LENGTH];
}BlocoMinerado;
```

  OBS.: Se você for aluno de ED1, a struct acima deverá ser transformada em uma struct auto-referenciada para permitir a inserção de um bloco minerado em uma lista encadeada de sua escolha. Nesta lista, um único nó registrará os dados do campo bloco bem como de seu hash válido.

4 Parâmetros de Simulação da BlockChain

- Quantidade de endereços do sistema: 256 (de 0 até 255);

- Valor da recompensa para o minerador: 50 bitcoins;
- Total de blocos a serem minerados: 1000 blocos (alunos de ED1) e 30000 (alunos de ED2);
- Forma de armazenamento dos blocos: lista encadeada de sua escolha (ED1), arquivo (ED2). No caso de ED2, a escrita no arquivo deve ocorrer em grupos de 16 blocos minerados de forma. Crie um vetor de blocos minerados de 16 posições, preencha-o e grave-o no arquivo.
- Quantidades de transações por bloco: Mínima: 0, Máxima: 61. Essa quantidade deve ser gerada aleatoriamente conforme seção 5.
- Quantidade máxima de bitcoins de uma transação: Mínimo: 0, Máximo: Total de bitcoins do endereço de origem escolhido para a transação. O endereço de origem de uma transação é definido aleatoriamente dentre os endereços que possuem bitcoin (ver subseção a seguir).

4.1 Como Gerar os Valores das Transações

- **Transações no Bloco Gênesis**: O bloco número 1 (chamado gênesis), não contêm transações. Neste caso, o campo data deverá ser preenchido com a string "*The Times 03/Jan/2009 Chancellor on brink of second bailout for banks*" (sem aspas). As demais posições do vetor data deverão conter o valor zero e a última posição (onde está o código do minerador), deverá conter o endereço do minerador definido aleatoriamente (conforme seção 5).
- **Transações nos Demais Blocos**: A partir do bloco 2, os três números de uma transação devem ser gerados aleatoriamente conforme seção 5. Contudo, somente poderão ser escolhidos como endereço de origem aqueles endereços que possuirem bitcoins. Qualquer endereço de destino pode ser escolhido. Além disso, a quantidade inteira de bitcoins da transação deve ser aleatoriamente escolhida respeitando a quantidade de bitcoins do endereço de origem escolhido. Para exemplificar melhor, considere o seguinte exemplo. Se o endereço 125 minerou o bloco gênesis, então ele terá 50 bitcoins de recompensa e todos os demais endereços terão zero. Assim, todas as transações do bloco dois só poderão ter o 125 como endereço de origem. Se 4 transações foram aleatoriamente definidas para o bloco 2 (qtd. máximad e transações), então outros 8 números deverão ser gerados aleatoriamente: 4 representando os endereços de destino e 4 representando as quantidades de bitcoins de cada transação. Como nesse caso o máximo de bitcoins do endereço de origem é 50, este será o maior valor aleatório que poderá ser escolhido. Suponha que 30 bitcoins sejam escolhidos já para a primeira transação. Então, a quantidade máxima da segunda será 20. Se 20 for aleatoriamente definido para a segunda, então as duas transações restantes devem conter zero bitcoins cada uma. Por fim, a partir do terceiro bloco, todos os endereços de destino que receberam bitcoins na segundo bloco, poderão ser escolhidos como endereços de origem.

5 Como Gerar os Número Aleatórios

Todos os números aleatórios do projeto deverão ser gerados pela função `genRandLong` da biblioteca disponível em `https://github.com/ESultanik/mtwister`. Em todas as chamadas desta função, deverá ser passada a mesma variável do tipo `MTRand` inicializada com o valor (semente) `1234567`.