Lista de Exercícios 1 - AEDS 3

Henrique Oliveira da Cunha Franco

1 Conceitos Básicos / Organização de Campos e Registros

1.1 Q1

A menor quantidade de bytes lida ou escrita em um disco rígido por operação realizada é a mesma do tamanho do setor do disco, que tipicamente é de 4.096 bytes, pois o setor é a menor unidade de dados que pode ser lida ou escrita em um arquivo.

1.2 Q2

O termo "cilindro"em um disco rígido refere-se a uma estrutura geométrica imaginária que compreende todas as trilhas de um disco que estão na mesma posição radial em cada uma das superfícies de gravação. Em outras palavras, é uma coleção de trilhas que estão alinhadas verticalmente uma sobre a outra, formando um cilindro imaginário.

1.3 Q3

Os arquivos são gravados cilindro a cilindro em discos rígidos devido à eficiência de acesso, já que aos cabeçotes se moverem somente entre trilhas de um cilindro, evita-se o tempo de reposicionamento radial, otimizando o desempenho de leitura/gravação e minimizando a latência.

1.4 Q4

Os componentes do tempo de acesso em discos rígidos envolvem;

- Tempo de bsuca (seek time): É o tempo que o cabeçote de leitura/gravação leva para se mover e posicionar-se no cilindro correto. Em média, corresponde a 2/3 do tempo máximo.
- Latência rotacional: É o tempo que o disco leva para girar até que o setor desejado esteja sob o cabeçote de leitura/gravação. Em média, corresponde a 1/2 da rotação.
- Tempo de transferência: É o tempo necessário para transferir os dados entre o disco e o computador uma vez que o cabeçote esteja posicionado corretamente. Depende da velocidade de rotação do disco e da densidade de dados da mídia.

$$\text{Taxa transferência} = \frac{\text{nº setores/trilha * tamanho do setor}}{\text{60 / velocidade em RPM}}$$

  Eventualmente é necessário considerar o tempo de mudança de cabeçote e o tempo de mudança de cilindro.

1.5 Q5

As três características que definem bom identificador de entidades em arquivos indexados são as seguintes:

1. **Único e Não Ambíguo:** Cada entidade deve ter um identificador único, não reutilizável e inequívoco. Isso evita conflitos e garante que cada entidade possa ser facilmente identificada de forma exclusiva. A unicidade é crucial para evitar duplicações ou confusões nos dados.

2. **Compacto e Eficiente:** O identificador deve ocupar o menor espaço possível em termos de armazenamento e processamento. Isso é importante para minimizar o uso de recursos do sistema e melhorar o desempenho, especialmente em grandes conjuntos de dados. Um identificador compacto também facilita a indexação e a busca rápida das entidades.

3. **Estabilidade e Persistência:** O identificador deve permanecer inalterado ao longo do tempo e ser persistente em diferentes operações de manipulação de dados, como atualizações ou exclusões. Isso garante a consistência e integridade dos dados, permitindo que outras entidades ou sistemas dependam consistentemente desse identificador para referência ou associação.

1.6 Q6

Para armazenar a data e hora de um evento em um arquivo binário de forma eficiente, podemos representá-los como timestamps Unix, utilizando a contagem de dias desde a época Unix para a data e o número de segundos ou milissegundos desde a meia-noite para a hora. Isso requer apenas 8 bytes no total (4 bytes para a data e 4 bytes para a hora) e proporciona uma representação compacta e precisa da informação temporal. No caso, na linguagem Java, esses 8 bytes também poderiam ser agrupados em uma única variável do tipo long. Por sua vez, esses valores podem ser armazenados diretamente no arquivo binário, economizando espaço e garantindo a facilidade de manipulação e processamento dos dados de data e hora.

1.7 Q7

Para tomar proveito do problema dos registros lápide, é possível realizar o seguinte; inserir o id do último registro excluído ao início do arquivo binário, e, assim que for feita uma nova inserção, conferir se é possível incluí-lo diretamente no lugar do registro lápide, assim, substituindo-o. Caso o espaço do registro lápide for menor do que o espaço necessário para ser substituído pelo novo registro, apenas insira o novo registro ao final do arquivo e mantenha o id do registro lápide ainda guardado.

1.8 Q8

A estrutura de registro de tamanho variável seria feita com base nos seguintes itens:

- Tamanho do Registro (2 bytes) - short: Este campo indicará o tamanho de todo o registro em bytes, permitindo uma leitura eficiente dos registros.
- ID (4 bytes) - inteiro: Serão utilizados 4 bytes para armazenar os ID's.
- Nome (variável) - string: O nome será armazenado como uma sequência de caracteres, utilizando o número de bytes indicado no campo anterior.
- Data de Nascimento (4 bytes) - int: Serão utilizados 4 bytes para armazenar a data de nascimento no formato dia/mês/ano, tendo como base a timestamp Unix.
- Gênero (1 byte) - char: Um byte é suficiente para armazenar o gênero.
- Cidade (variável) - string: O nome da cidade será armazenado como uma sequência de caracteres, utilizando o número de bytes indicado no campo anterior.
- UF (2 bytes) - string: Serão utilizados 2 bytes para armazenar o UF, assumindo que não serão usados mais do que 255 UFs diferentes.

1.9 Q9

As vantagens de utilização de uma variável do tipo int ou long em relação a uma variável do tipo string são listadas a seguir;

- **Maior facilidade de manipulação das datas**
- **Compactação e armazenamento mais bem administrado**
- **Capacidade de maior precisão na hora de se referir às datas**

A variável int pode ser utilizada quando se deseja que seja especificado um intervalo de tempo, ou seja, um período menos específico e com uma janela temporal um pouco menor. Baseando-se na timestamp Unix, é possível representar qualquer data com uma boa precisão até 19 de Janeiro de 2038. Já, uma variável long poderá representar datas além do ano 2038, acomodando assim programas que precisam se acomodar com uma faixa de datas muito mais ampla, o que acaba tornando-a mais adequada para sistemas que exigem uma maior flexibilidade temporal.

1.10 Q10

Sim, utilizando um campo que guarda o valor máximo de bytes que pode ser reservado para um outro atributo do registro. Um exemplo seria um sistema que requer o armazenamento do nome de uma pessoa. Ao ler o tamanho do registro, o programa reserva um número de bytes específico para tal, e realiza o mesmo para cada campo presente no registro. Assim, o atributo nome, bem como o próprio registro inteiro, terá um tamanho fixo próprio.

1.11 Q11

A estrutura de registro de tamanho variável seria feita com base nos seguintes itens:

- Tamanho do Registro (2 bytes) - short: Este campo indicará o tamanho de todo o registro em bytes, permitindo uma leitura eficiente dos registros.
- Código (4 bytes) - inteiro: Serão utilizados 4 bytes para armazenar os códigos.
- Comprimento da Descrição (1 byte) - unsigned byte: Para indicar o comprimento da descrição da infração, podemos usar o tipo byte. Como o comprimento da descrição provavelmente não excederá 255 caracteres, um byte é suficiente para representar esse valor.
- Infração (variável) - string: A descrição referida à infração cometida será armazenada como uma sequência de caracteres.
- Pontos (1 byte) - short: Apenas um único byte será suficiente para armazenar o número de pontos, já que seu valor máximo é 40.
- Valor (4 bytes) - float: Serão utilizados 4 bytes para armazenar o valor, já que é necessário ter a precisão fracionária fornecida por variáveis do tipo float.

1.12 Q12

Os campos de um registro USUÁRIO envolveria os seguintes atributos;

- E-mail (variável) - string: o email do usuário é guardado como uma sequência de caracteres.
- Senha (variável) - string: o email do usuário é guardado como uma sequência de caracteres alfanuméricos.
- Notas de Filmes (variável) - array/lista de shorts: Esse atributo agirá como registro das notas que o usuário deu a diferentes filmes. Pode ser implementado como um array ou lista, onde cada elemento contém um ID hipotético do filme e a nota atribuída pelo usuário.

1.13 Q13

Serão utilizados 12 bytes para armazenar a string EDUCAÇÃO nesse sistema, sendo 1 deles gasto para cada caractere (exceto Ã e Ç que gastam 2 cada) + 2 bytes para o número que guarda o tamanho do registro (variável do tipo short).

1.14 Q14

Não, um arquivo sequencial não é necessariamente ordenado pois a sequencialidade de um arquivo se refere apenas à forma como os dados são organizados no disco, onde os registros são armazenados sequencialmente um após o outro.

1.15 Q15

Uma grande vantagem adquirida por meio da utilização da lápide é da pouca manutenção necessária sobre o processo de exclusão de registros. Caso contrário, seria necessário reorganizar o arquivo, movendo uma grande parcela de registros, tendo assim um custo alto ao realizar a exclusão do registro. Já, uma desvantagem é o possível gasto elevado com espaço, já que mesmo que o registro seja considerado pelo sistema como "lápide", ele ainda permanecerá da mesma maneira no arquivo, gastando assim os bytes que poderiam ser liberados caso ele tivesse sido, de fato, deletado.

1.16 Q16

O cabeçalho de um arquivo de dados é uma seção inicial do arquivo que contém informações sobre a estrutura e organização dos dados armazenados no arquivo, sendo geralmente utilizado para fornecer informações importantes que ajudam na interpretação e manipulação dos dados. Um exemplo de informação que ele pode guardar seria o nome de cada um dos campos dos registros, podendo estar junto ou não dos tamanhos reservados para cada um dos campos.

1.17 Q17

O melhor tipo de dados para uso como ID de clientes nesse sistema é uma variável do tipo int, que se trata de um valor compacto e que tem alta capacidade numérica.

1.18 Q18

Os campos lápide em um arquivo servem para evidenciar se um registro deve ser visto como excluído pelo sistema, mesmo que seus bytes ainda estejam presente no arquivo. Logo, supondo que a lápide é do tipo boolean, seria possível dizer que se um registro x possui seu atributo lápide como *true*, ele foi deletado, e se sua lápide estiver como *false*, ele ainda se encontra vivo/ativo.

1.19 Q19

Se o registro for menor que o espaço disponível, resultando em sobra de bytes no espaço reaproveitado, a ação a ser tomada dependerá da política de gerenciamento de espaço adotada pelo sistema. Existem algumas opções comuns que podem ser consideradas:

- **Compactação do Espaço Reaproveitado:** Após inserir o novo registro e identificar que há espaço não utilizado restante, o sistema pode realizar uma operação de compactação para remover esse espaço não utilizado. Isso envolve mover os registros subsequentes para ocupar o espaço vazio, eliminando assim a sobra de bytes.
- **Manutenção do Espaço Não Utilizado:** Em vez de compactar o espaço, o sistema pode optar por manter o espaço não utilizado para uso futuro. Isso pode ser útil se houver uma expectativa de que registros maiores possam ser inseridos posteriormente, permitindo um uso mais eficiente do espaço disponível.

- **Marcação do Espaço Não Utilizado:** O sistema pode marcar o espaço não utilizado como livre ou disponível para reutilização futura, mas mantê-lo separado dos registros ativos. Isso facilita a identificação e utilização desse espaço quando necessário, sem afetar a integridade dos registros existentes.

1.20 Q20

As opções disponíveis de implementação que podem ser realizadas para lidar com o espaço remanescente do registro devem ser:

- A movimentação dos registros subsequentes para acomodar o registro sem sobra de espaço;
- A marcação do espaço livre para ser ignorado e consequentemente deletado caso ocorra uma eventual ordenação no arquivo;
- A manutenção do espaço para caso sejam feitas futuras atualizações do registro que farão com que ele ocupe mais espaço.

2 Ordenação Externa

2.1 Q1

Antes de iniciarmos as intercalações, cada segmento terá no máximo 5 elementos ordenados. Considerando que cada segmento é preenchido até a capacidade máxima de ordenação antes de serem realizadas as intercalações, conclui-se que cada segmento estará cheio e pronto para a intercalação quando atingir essa capacidade máxima de elementos.

2.2 Q2

5.000 entidades e 25 entidades em memória principal

$$25.3^x = 5000$$

Logo, x = 5.

2.3 Q3

Permitindo segmentos de tamanho variável, o algoritmo pode ajustar dinamicamente o tamanho dos segmentos baseado na forma como os elementos estavam ordenados previamente, fazendo com que seja possível economizar algumas intercalações em alguns casos, e assim reduzindo o custo de realizar a ordenação.

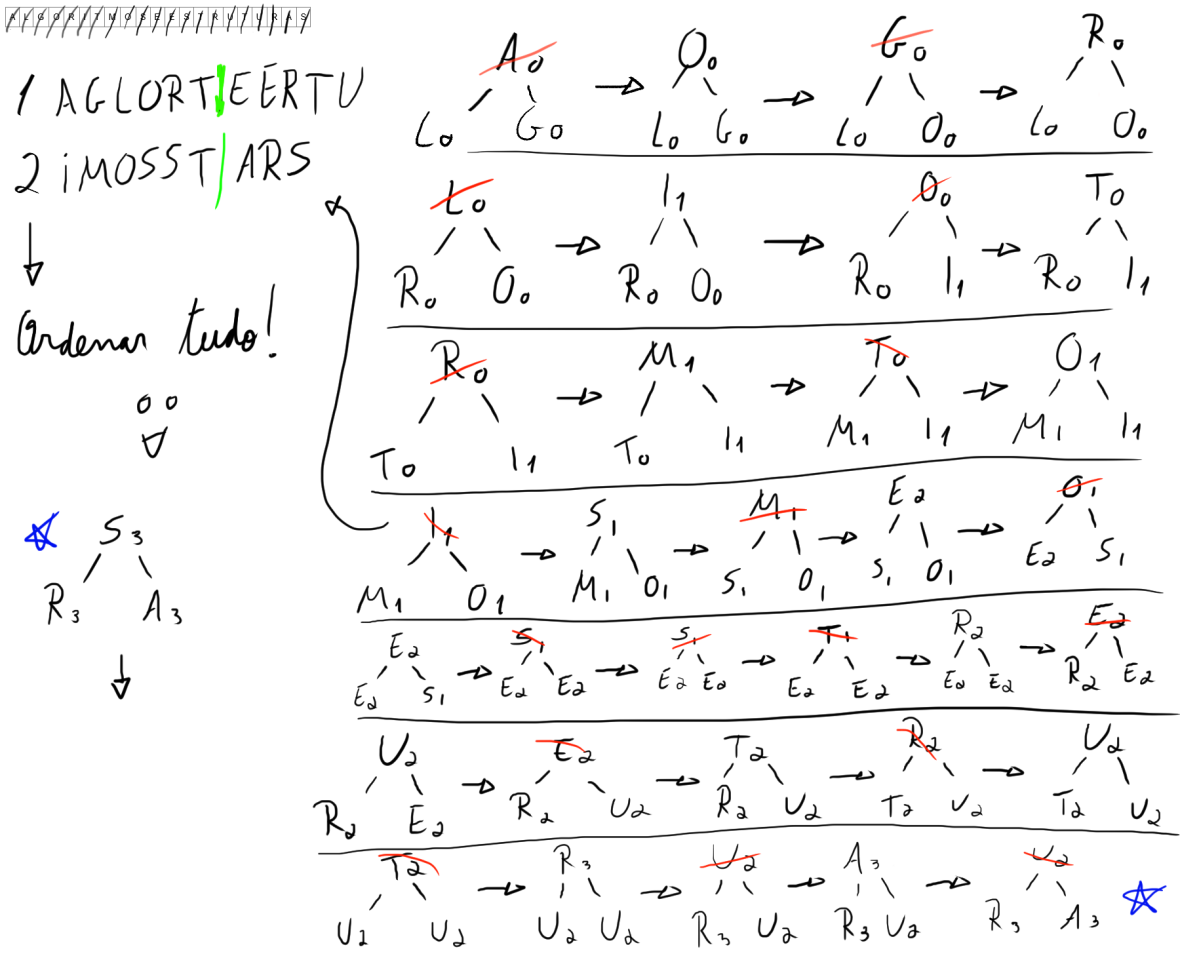
2.4 Q4

O principal benefício é conseguir criar blocos maiores de entidade maiores utilizando uma quantidade de bytes da memória principal, fazendo com que possam ser feitos blocos maiores, fazendo com que sejam economizadas intercalações, e haja uma utilização mais eficiente da memória principal na fase de distribuição.

2.5 Q7

O benefício principal é conseguir criar blocos maiores de entidade maiores utilizando uma quantidade de bytes da memória principal, fazendo com que possam ser feitos blocos maiores, fazendo com que sejam economizadas intercalações, e haja uma utilização mais eficiente da memória principal na fase de distribuição.

2.6 Q9



3 Arq. Indexados / Árvores B, B+ e B*

3.1 Q4

A vantagem do uso de arquivos indexados é que há uma redução considerável no custo para realização de buscas ou outras ações que necessitem de alguma busca nelas. A desvantagem é o aumento na complexidade do programa e também o aumento no custo de inserções e remoções em alguns casos pois é necessário que o arquivo de índice seja alterado, que pode ser custoso as vezes. Além disso, o arquivo de índice ocupa espaço na memória, fazendo com que o conjunto dos dados do projeto sejam ligeiramente mais pesados.

3.2 Q5

Para reaproveitar o espaço desperdiçado, é possível, na inserção, percorrer todos elementos do arquivo de dados até encontrar algum desses espaços e depois inserir o elemento novo nesse lugar caso caiba, mas mantendo o campo que guarda o tamanho do registro com o valor anterior.

3.3 Q6

Um índice secundário é um índice que a ordem dos elementos não é a mesma do arquivo de dados principal. Um índice direto é aquele que armazena o endereço dos elementos no arquivo de dados

principal, e não um endereço de um dado em algum outro índice. Um índice denso é aquele que contém todos os elementos do arquivo principal ao qual faz referência.

4 Hashing

4.1 Q1

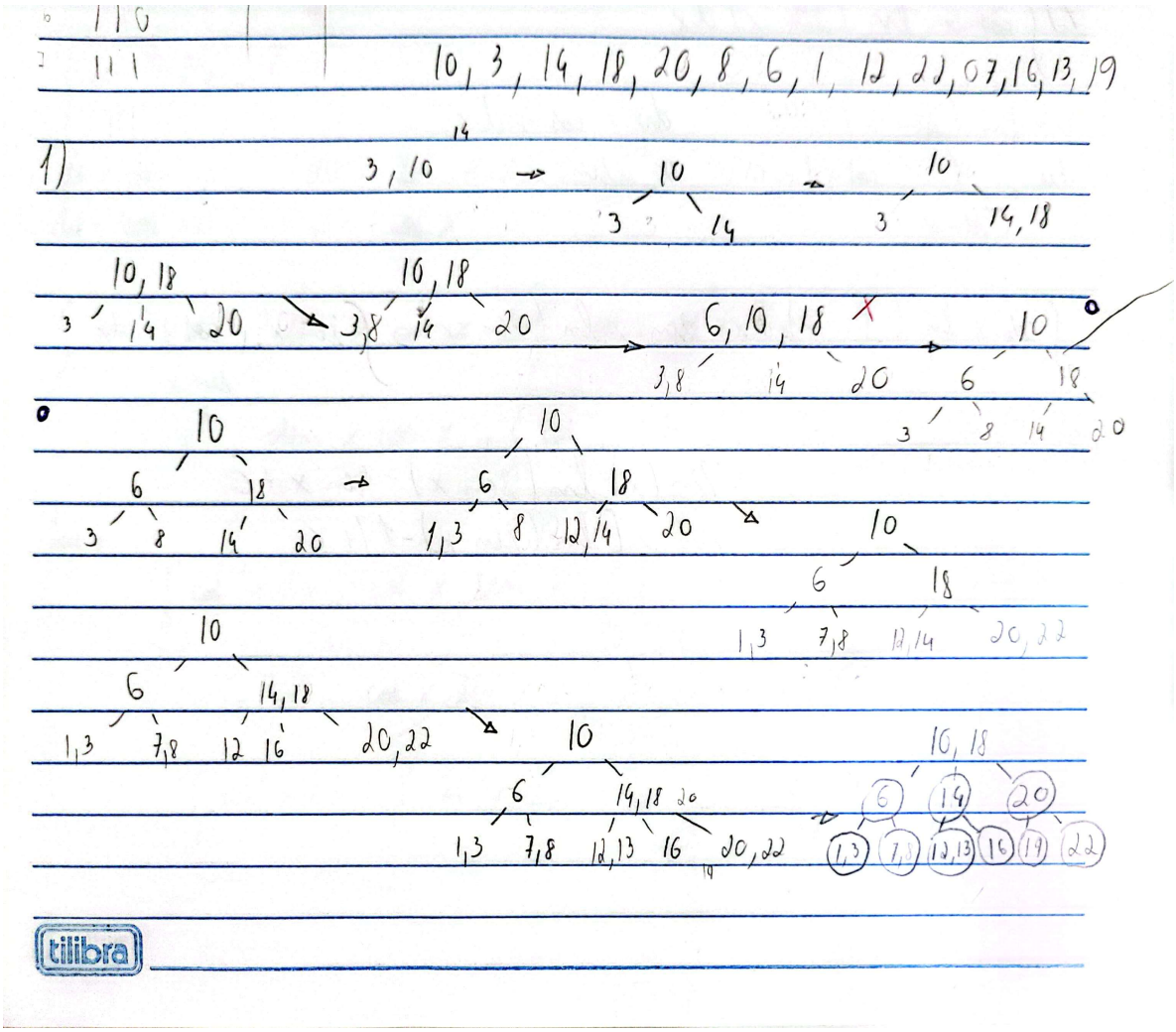
A vantagem do uso de buckets para tabelas hash em disco é que é possível utilizar de forma mais eficiente os setores do disco rígido, que contém 4096 bytes, fazendo com que utilizar buckets de tamanhos maiores faça com que seja possível ler vários elementos de uma vez sem haver um tempo de busca e latência rotacional, acelerando a busca na maior parte dos casos.

4.2 Q2

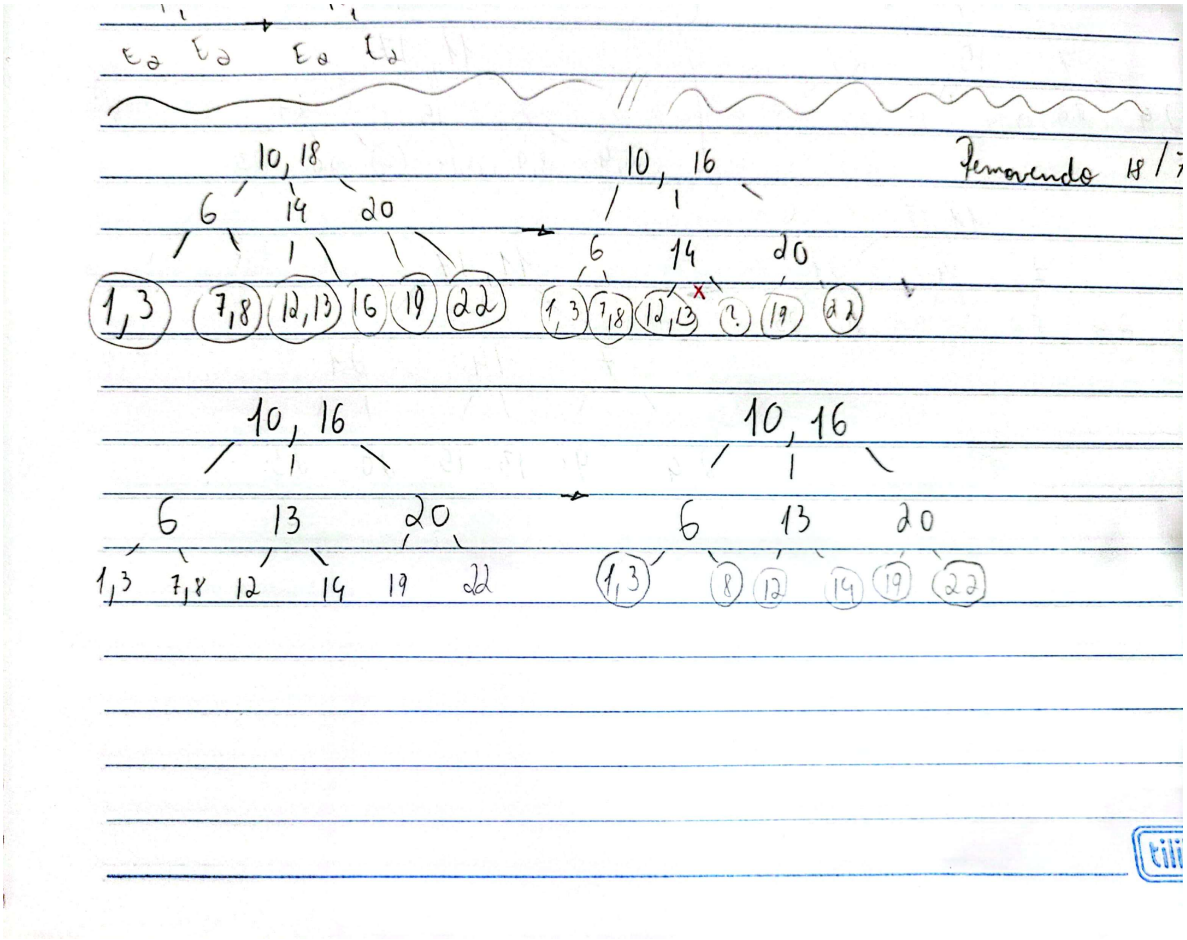
A partir da profundidade de certo diretório de um hash extensível, é possível calcular qual é o número que a conta x%2p̂ utiliza.

5 Todo conteúdo

5.1 Q1 a.



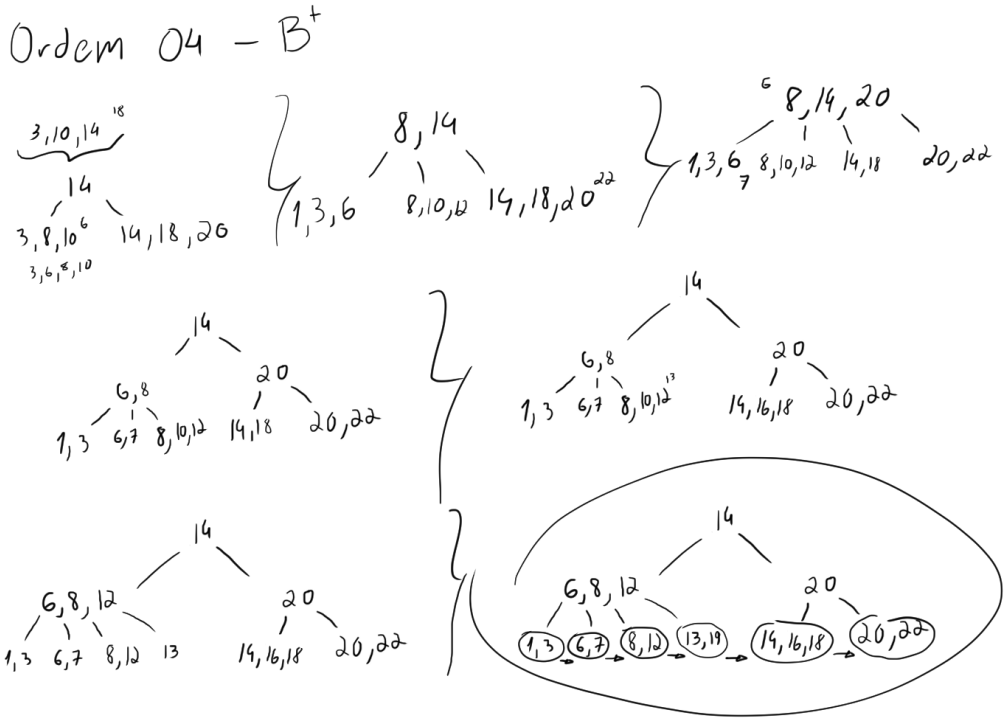
Parte 1 da questão



Parte 2 da questão

1 - Suponha que as seguintes chaves foram inseridas em uma estrutura de pesquisa: 10 / 06 / 14 / 18 / 20 / 08 / 06 / 01 / 12 / 22 / 07 / 16 / 03 / 19 (nesta ordem) e em seguida foram removidas as chaves 18 e 07 (nesta ordem).

a) Pede-se que se represente a estrutura resultante caso seja utilizada Árvore B de ordem 03.
b) Pede-se que se represente a estrutura resultante caso seja utilizada Árvore B+ de ordem 04.



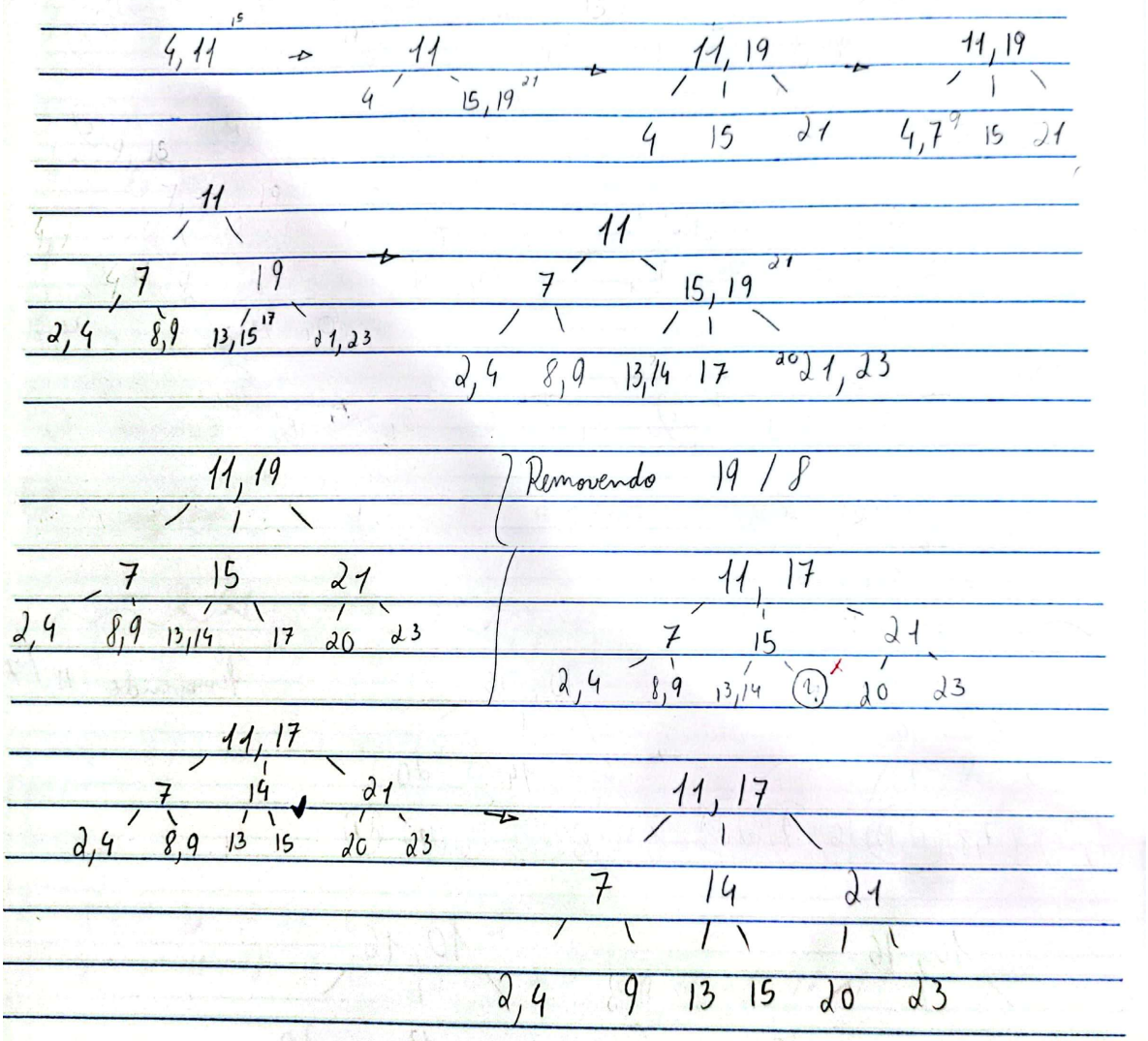
5.2 Q1 b.

5.3 Q2 a.

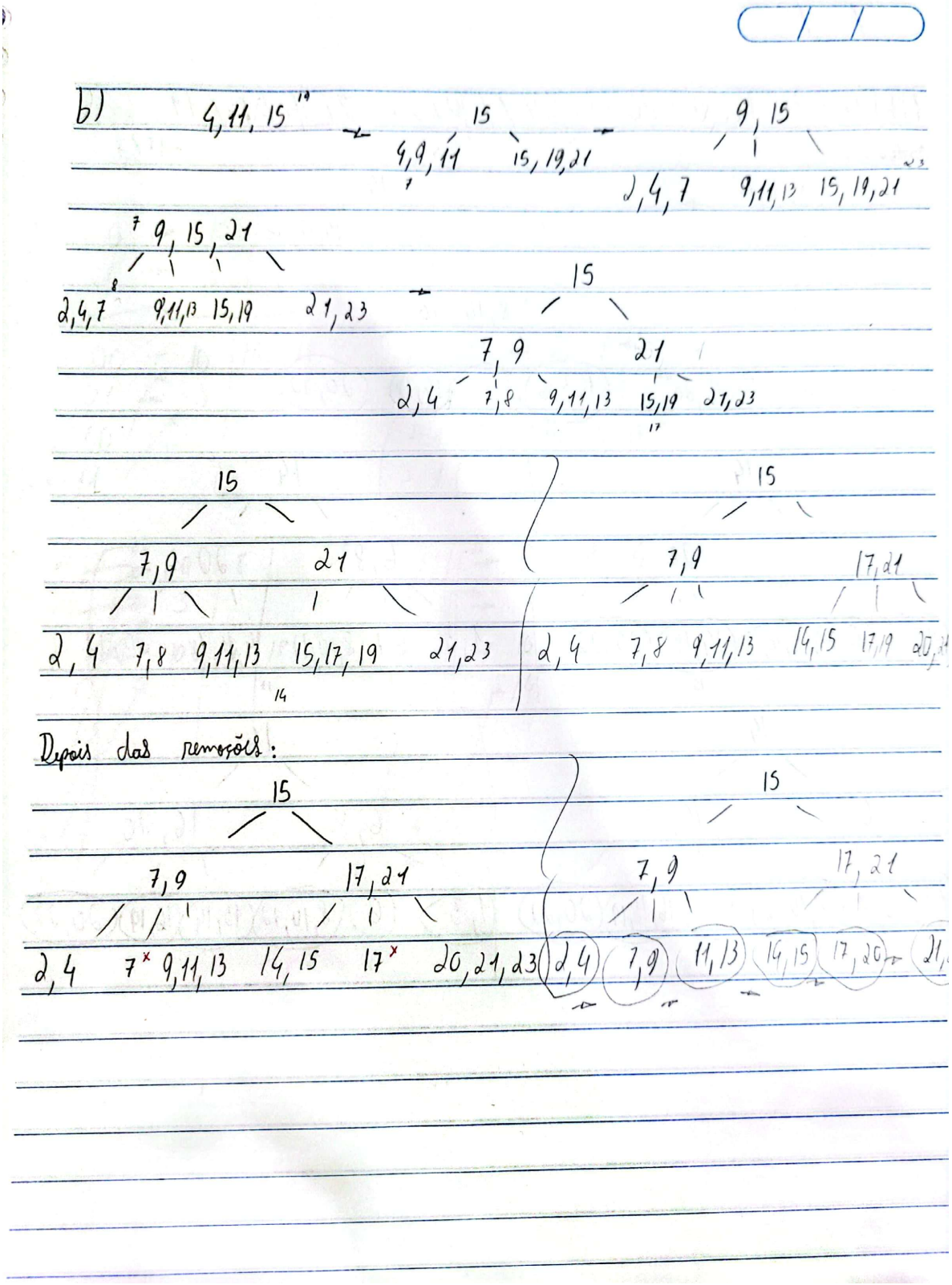
5.4 Q2 b.

2) 11 / 4 / 15 / 19 / 21 / 9 / 7 / 2 / 13 / 23 / 8 / 17 / 14 / 20

a) Ordem 3



Removendo 19 / 8


b)
4, 11, 15
15
4, 9, 11
15, 19, 21
9, 15
2, 4, 7
9, 11, 13
15, 19, 21
7 9, 15, 21
2, 4, 7
9, 11, 13
15, 19
21, 23
15
7, 9
21
2, 4
7, 8
9, 11, 13
15, 19
21, 23
15
7, 9
21
2, 4
7, 8
9, 11, 13
15, 17, 19
21, 23
14
15
7, 9
17, 21
2, 4
7, 8
9, 11, 13
14, 15
17, 19
Depois das remoções:
15
7, 9
17, 21
2, 4
7
9, 11, 13
14, 15
17
20, 21, 23
15
7, 9
17, 21
2, 4
7, 9
11, 13
14, 15
17, 20