Respostas - Tabela Hash com Sondagem Linear

Aluno: José Francisco da Silva Neto

1.Por que precisamos garantir que o resultado de hashCode() seja transformado em um índice não-negativo e dentro do tamanho da tabela? O que aconteceria se você usasse hashCode() diretamente como índice?

hashCode() pode ser negativo e ter qualquer valor dentro da faixa de Integer. Se usássemos diretamente como índice, teríamos dois problemas: índices negativos causariam exceção ArrayIndexOutOfBoundsException ao acessar o vetor, e o valor poderia ser muito maior que a capacidade, também gerando erro.

Para evitar isso, transformamos o hashCode() em um índice válido: se a chave for nula, usamos 0; senão, chamamos hashCode(), usamos Math.abs() para garantir que seja não-negativo, e aplicamos o operador % capacidade. Assim, o resultado sempre fica entre 0 e capacidade − 1. Sem essa transformação, correríamos o risco de acessar posições inválidas do vetor, causando erro em tempo de execução e corrompendo a estrutura da tabela.

2.Ao inserir várias chaves que colidem entre si (por exemplo: 10, 21, 32, 43), como a sondagem linear distribui essas chaves pelo vetor? Descreva o que ocorre passo a passo quando há colisão.

Quando várias chaves produzem colisão, como 10, 21, 32, 43, 54, 65 com capacidade 11, a sondagem linear age assim: primeiro, calcula o índice inicial da chave. Se a posição estiver livre, a chave é inserida ali. Se já houver outra chave diferente naquele índice, acontece a colisão; então, a tabela não desiste, apenas anda para a próxima posição, somando 1 ao índice e usando o operador % capacidade para não sair do vetor.

Se essa nova posição também estiver ocupada, o processo continua avançando células uma a uma até encontrar uma posição vazia ou marcada como removida. Na prática, as chaves que colidem vão sendo colocadas em posições consecutivas, formando um "bloco" contínuo de posições ocupadas. Por exemplo, se 10 vai para

posição 10, e 21 colide, 21 vai para 0, e 32 também colide com ambas, vai para 1, e assim por diante.

3.Por que a busca precisa percorrer as mesmas posições, na mesma ordem, que a inserção percorreu? O que aconteceria se ela percorresse de forma diferente?

A busca na sondagem linear precisa seguir exatamente o mesmo caminho que a inserção seguiu, começando do mesmo índice inicial e avançando as posições na mesma ordem. Isso é essencial porque a localização final da chave depende da sequência de colisões que aconteceu quando ela foi inserida.

Se a busca usasse outra ordem, por exemplo, pulando índices ou sondando em outro padrão, poderia deixar de passar pela posição onde a chave realmente foi armazenada. Nesse caso, a busca retornaria que a chave "não existe", mesmo ela estando na tabela, o que tornaria a estrutura incorreta e inutilizável.

4.Por que, durante uma busca, encontrar uma posição com estado VAZIO significa que a chave procurada não está na tabela? Explique o motivo lógico por trás disso.

Durante a busca, se a sondagem chegar a uma posição com estado VAZIO, isso significa que aquela célula nunca recebeu nenhuma chave desde a criação (ou desde a última reconstrução) da tabela.

Na sondagem linear, a inserção só anda para frente quando encontra colisões em posições já ocupadas. Se, seguindo o caminho natural da chave, encontramos uma posição que nunca foi usada, fica claro que nenhuma inserção continuou além desse ponto para essa chave. Por isso, logicamente é impossível que a chave esteja em alguma posição posterior na sequência da sondagem, e a busca pode terminar concluindo que a chave realmente não está na tabela.

5.Por que a sondagem NÃO pode parar quando encontrar uma posição REMOVIDO? Qual é a importância desse estado intermediário?

A sondagem não pode parar ao encontrar uma posição com estado REMOVIDO porque esse estado indica que já existiu uma chave ali, mas que foi removida

depois. Isso quer dizer que, no passado, a cadeia de sondagem para alguma chave passou por ali e continuou para as posições seguintes.

Assim, podem existir chaves mais adiante que dependem dessa continuidade de busca. Se a busca parasse ao ver REMOVIDO, a cadeia seria interrompida artificialmente, e chaves que foram inseridas depois de uma colisão nesse ponto se tornaram invisíveis para futuras buscas. O estado REMOVIDO, portanto, funciona como um "marco" dizendo: "já houve algo aqui, continue procurando".

6.O que acontece na inserção quando você insere uma chave que já existe na tabela? Por que isso é importante para manter a consistência da estrutura?

Quando inserimos uma chave que já existe na tabela, a sondagem linear percorre o vetor exatamente como na busca e, ao encontrar a posição onde aquela chave está armazenada, não cria uma nova entrada; em vez disso, apenas atualiza o valor associado à chave. Na prática, a mesma posição continua representando a mesma chave, mas com um valor atualizado.

Isso é importante para manter a consistência: numa tabela hash, cada chave deve ter no máximo um valor associado. Se, a cada inserção de uma chave repetida, fosse criada uma nova posição, teríamos duplicidade de chaves, o que quebraria o modelo de mapa chave → valor e causaria buscas incorretas.

7.Por que não podemos simplesmente marcar uma posição como VAZIO quando removemos uma chave? Qual problema isso causaria nas futuras buscas?

Não podemos simplesmente marcar uma posição como VAZIO quando removemos uma chave porque VAZIO significa "nunca houve nada aqui". Se fizéssemos isso, a busca poderia terminar antes da hora.

Por exemplo: uma chave A foi inserida, causou colisão, e por isso uma outra chave B foi colocada mais à frente. Se removermos A e colocarmos VAZIO na sua antiga posição, ao buscar por B a sondagem pode bater nesse VAZIO e concluir que "a partir daqui nunca teve nada, então B não existe", mesmo B estando logo depois. O estado REMOVIDO resolve isso: indica que ali havia uma chave, mas ainda permite

que a sondagem continue procurando por outras chaves que foram colocadas à frente.

8.Como o fator de carga influencia o desempenho da sondagem linear? Quando o fator de carga está muito alto, o que acontece com o tempo de busca e inserção?

O fator de carga α é a razão entre o número de elementos e a capacidade da tabela, medindo quão cheia está a tabela.

Quando o fator de carga é baixo (a tabela tem muitas posições vazias), as cadeias de sondagem tendem a ser curtas, e inserções, buscas e remoções costumam ser rápidas, próximas de tempo constante O(1).

Já quando o fator de carga está alto (a tabela quase cheia), tornam-se comuns longas sequências de posições ocupadas, os chamados clusters. A sondagem precisa percorrer muitos elementos antes de encontrar uma posição vazia ou a chave desejada, fazendo com que o tempo de busca e de inserção aumente bastante, podendo se aproximar de um tempo proporcional ao tamanho da tabela, ou seja, O(n) no pior caso.

9.O que acontece quando o algoritmo percorre todas as posições e não encontra lugar para inserir? Por que isso significa que a tabela está "cheia", mesmo que existam posições REMOVIDO? (Pense sobre como o método garante a busca correta.)

Quando o algoritmo de inserção percorre todas as posições possíveis na tabela e não encontra nenhum lugar apropriado para inserir (ou seja, não acha nenhuma posição marcada como VAZIO ou REMOVIDO), isso significa que, para aquele esquema de sondagem e para aquela capacidade, a tabela está cheia.

Pode parecer estranho se ainda existem posições que já foram usadas e marcadas como REMOVIDO. Porém, como a sondagem depende de garantir que busca e inserção passam pelas mesmas posições, não é possível apenas "pular" alguma posição para forçar uma inserção; se não há célula válida nessa sequência, a tabela

está saturada para aquele conjunto de chaves e não pode mais acomodar novas chaves sem quebrar a garantia de que buscas sempre encontram o que foi inserido.

10.Compare brevemente a sondagem linear com o encadeamento separado:

- **em qual delas o pior caso de busca tende a ser mais grave?**
- **qual delas lida melhor com fatores de carga altos?**
- **qual delas facilita mais a remoção?**

Na sondagem linear, com fator de carga alto, cadeias longas podem tornar a busca quase uma varredura completa do vetor, com pior caso O(n) e clusters grandes. No encadeamento separado, o pior caso também é O(n) (todas as chaves no mesmo bucket), mas na prática uma função de hash razoável tende a distribuir bem os elementos entre os buckets, tornando esse cenário menos frequente.

Em relação a fatores de carga altos, o encadeamento separado lida melhor, porque a tabela pode continuar funcionando mesmo quando o número de elementos ultrapassa a capacidade do vetor principal; o custo cresce pelo tamanho das listas, não por "encher" o vetor. Na sondagem linear, fatores de carga elevados aumentam muito o tamanho das cadeias de sondagem.

Na remoção, o encadeamento separado é mais simples: basta remover o nó da lista ligada correspondente; não é necessário usar estados especiais. Na sondagem linear, a remoção exige cuidado com o estado REMOVIDO para não quebrar a lógica de busca.

11.Na sondagem linear acontece um fenômeno chamado “cluster primário” (várias células ocupadas consecutivamente). Você percebeu esse comportamento em seus testes? Descreva o que viu ao inserir várias chaves que colidem.

Na sondagem linear ocorre o fenômeno chamado cluster primário: quando várias chaves que colidem vão sendo inseridas, elas acabam ocupando uma sequência de posições consecutivas no vetor.

Nos testes com chaves como 10, 21, 32, 43, 54, 65, é comum ver isso acontecer. A primeira chave entra no índice calculado normalmente. A próxima, ao colidir, é empurrada para a próxima posição livre. A seguinte, ao também colidir com as mesmas regiões, entra em outra posição logo em seguida, e assim por diante. O resultado é um bloco de células ocupadas, sem espaços vazios entre elas, o que forma um cluster. Esse cluster, com o tempo, tende a crescer e a atrair ainda mais colisões naquela região.

12.Se a capacidade da tabela é fixa, o que você faria para evitar que ela fique cheia? O que muda quando você redimensiona a tabela? (Pense sobre recalcular índices e copiar elementos.)

Se a capacidade da tabela é fixa, uma forma de evitar que ela fique cheia e que o desempenho caia é fazer o redimensionamento quando o fator de carga atinge um limite escolhido, por exemplo, 0,5 ou 0,7.

O processo de redimensionamento normalmente envolve criar um novo vetor maior (por exemplo, o dobro do tamanho anterior), redefinir a capacidade e reinserir todas as chaves na nova tabela. Ao reinserir, como a capacidade mudou, os índices calculados por hash capacidade também mudam, o que redistribui os elementos e quebra clusters grandes.

Esse processo é chamado de rehash e melhora o desempenho das operações futuras às custas de um custo mais alto naquele momento de redimensionamento. Todos os índices mudam porque o cálculo hash % capacidade passa a usar outro valor de capacidade, e é necessário copiar (na verdade, reinserir) todos os elementos na nova tabela para manter a correção da sondagem.

13.Em termos de complexidade, qual é o custo médio e o pior caso para:

- **inserção,**

- **busca,**
- **remoção**

na sondagem linear? Explique com base no comportamento da sondagem.

Em média, quando a tabela é bem dimensionada e o fator de carga não é muito alto, inserção, busca e remoção têm custo próximo de constante, O(1), porque em geral é preciso sondar apenas poucas posições para encontrar um lugar livre ou a chave procurada.

No pior caso, porém, quando há muitos elementos e clusters grandes, qualquer uma das três operações pode precisar percorrer quase todas as posições do vetor, o que significa um custo O(n), onde n é a capacidade da tabela.

Na prática, o objetivo é manter o fator de carga em um nível que deixe o comportamento próximo do caso médio, evitando que a tabela se torne muito cheia.

14.Depois de realizar seu teste:

Quantas colisões você percebeu ao inserir as chaves propostas?

Em qual posição a primeira colisão ocorreu?

Quantas posições a sondagem precisou avançar até encontrar um local livre?

Explique observando o seu próprio output da execução.

Sobre as colisões observadas no teste prático, é importante olhar a saída do próprio programa. Ao inserir as chaves 10, 21, 32, 43, 54, 65 em uma tabela de capacidade 11, algumas delas certamente caem em índices iguais ou próximos, causando colisões.

A primeira colisão é aquela em que, ao inserir uma chave, o índice inicial calculado já está ocupado, e a tabela precisa avançar para a próxima posição pela sondagem linear. A quantidade de posições avançadas até encontrar um local livre corresponde ao número de colisões sofridas por essa inserção específica.

No relatório, descreva com base no output: qual chave gerou a primeira colisão, qual era o índice inicial, qual posição estava ocupada e quantas vezes o algoritmo precisou somar 1 ao índice até conseguir inserir. Por exemplo: "Na inserção da chave X, o índice inicial foi Y, mas a posição já estava ocupada (por Z), então a sondagem avançou para Y+1, Y+2, ... até a posição W, onde finalmente inseriu."

15.O que aconteceu quando você removeu uma chave e, depois disso, buscou novamente essa chave? Explique o comportamento observado com base no estado REMOVIDO.

Quando uma chave é removida e, depois disso, fazemos uma busca por essa mesma chave, o comportamento esperado é que a busca não a encontre mais. Isso acontece porque, na remoção, a posição da chave é marcada como REMOVIDO, e a chave e o valor são apagados.

Em uma busca futura, a sondagem linear passará pelo local REMOVIDO, mas não verá mais a chave ali, então continuará procurando nas posições seguintes. Como a chave foi removida e não existe em nenhuma outra posição, a busca termina retornando null (ou algum indicador de ausência).

Ao mesmo tempo, graças ao estado REMOVIDO, outras chaves que foram colocadas à frente na mesma sequência de sondagem continuam sendo encontradas corretamente. Isso demonstra que REMOVIDO não bloqueia a sondagem para outras chaves da mesma cadeia, e que a chave removida não é mais encontrada, mas as chaves que colidiram com ela e foram inseridas depois continuam acessíveis.