QXD0115 - ESTRUTURA DE DADOS AVANÇADA - 01A - 2024.1

AC03 - envio

Fase de avaliação

Seu envio

Instruções para avaliação

Soluções

- **Questão 1.** Em uma tabela hash com tratamento de colisão por encadeamento exterior, o que acontece se, ao invés de usar listas duplamente encadeadas, você implementar os $buckets$ usando alguma outra estrutura de dados? Discuta as mudanças e identifique vantagens e desvantagens ao implementar a tabela hash usando cada uma das seguintes estruturas de dados de dados:

  1. array
  2. lista encadeada ordenada
  3. fila
  4. pilha
  5. lista duplamente encadeada

  Solução: Essa questão é subjetiva e, para avaliá-la, você precisa ler o que o seu colega escreveu e avaliar se faz sentido. Se ele tocou nos pontos importantes. A seguir segue uma breve discussão:
  Ao longo da discussão a seguir, supomos que a tabela hash contém n entradas. Consideraremos o pior caso, onde a tabela não é o redimensionada automaticamente e a função hash pode não ser muito boa.

  1. **Arrays:** Um array de tamanho fixo seria pior que uma lista duplamente encadeada, e esta não seria uma implementação de bucket viável.
     Por outro lado, um array dinâmico pode aumentar sempre que necessário. Nesse caso, procurar uma chave dependeria de busca linear e teria custo $O(n)$. Embora uma nova entrada pudesse simplesmente ser adicionada no final do array dinâmico, a um custo médio $O(1)$ e amortizado, primeiro precisamos ter certeza de que a chave não está no array. Logo, isso aumenta o custo de inserção para $O(n)$. Mas podemos fazer melhor se mantivermos o array ordenado. Então, nesse último caso, podemos usar o algoritmo de busca binária para ambas as operações (inserir e buscar), o que reduz o custo para $O(logn)$. Remoções de pares chave-valor poderiam ocorrer também em tempo $O(logn)$.
  2. **Lista encadeada ordenada:** Manter nossos buckets ordenados pelas chaves não ajuda muito. Ainda precisaríamos realizar uma busca linear para procurar uma chave ou inserir uma entrada, o que faz com que a complexidade do pior caso seja $O(n)$. Na prática, os buckets ordenados permitem interromper a pesquisa mais cedo se encontrarmos uma chave maior do que aquela que estamos procurando ou tentando inserir. No final, isso não afetaria a complexidade assintótica da pesquisa.
  3. **Fila:** Nesse tipo de implementação, quando formos procurar uma chave, no pior caso todas as n entradas estarão em uma fila e teríamos que retirar tudo da fila para encontrá-la. Assim, a pesquisa do pior caso custa $O(n)$. Inserir uma chave teria custo semelhante. O tempo de execução de usar uma fila não é muito diferente daquele da lista encadeada.
  4. **Pilha:**A análise que realizamos para filas aplica-se de forma idêntica às pilhas.
  5. **Lista duplamente encadeada:**Nós sabemos que, sob esta configuração, implementar buckets usando listas duplamente encadeadas gera um custo $O(n)$ no pior caso para procurar uma chave. Assim, para inserir e remover também custariam $O(n)$ no pior caso.

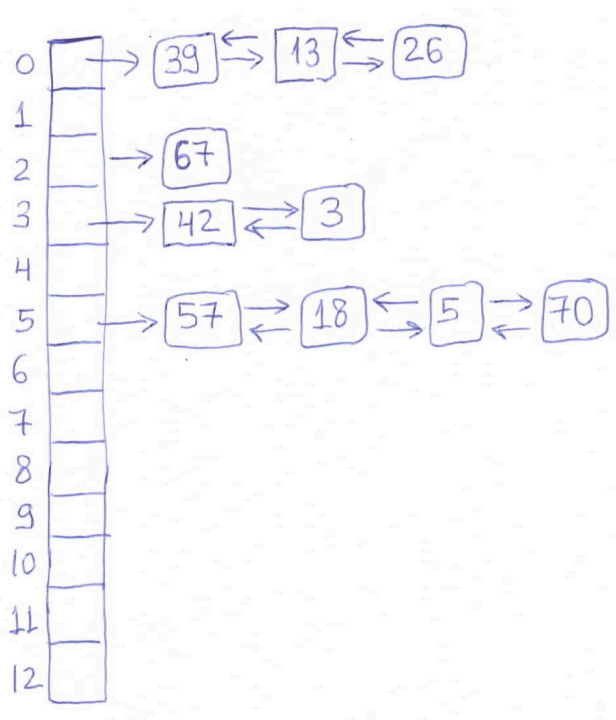
  Com base nesta análise, nossa melhor opção é usar um array dinâmico ordenado ou, melhor ainda, listas duplamente encadeadas para implementar os buckets da nossa tabela hash. A tabela hash deve empregar uma boa função hash e deve redimensionar a tabela subjacente quando o fator de carga atingir um valor constante (por exemplo, 1.0). Pelos teoremas vistos em sala, essas duas condições juntas garantem tempo de acesso $O(1)$ na média.

- **Questão 2.** Para essa questão, considere uma tabela hash que usa tratamento de colisão por encadeamento exterior. Suponha que estamos escrevendo uma função de hashing para strings de comprimento exatamente dois que consistem apenas nos caracteres de 'A' a 'Z'. Existem $26 \times 26 = 676$ strings diferentes. Ademais, suponha que sua aplicação usa apenas 79 dessas 676 possíveis palavras de duas letras. Sabendo disso, você ajustou sua tabela para ter tamanho $M = 100$ e você esperava implementar uma função de hashing que não causasse colisões na sua tabela. Porém você ainda vê colisões na maioria das vezes. Procure na internet, por exemplo na Wikipedia, sobre o **paradoxo do aniversário** e use-o para explicar esse fenômeno que está ocorrendo na sua tabela.

  Solução: Antes de tudo, note que você certamente teria colisão se o tamanho da sua tabela fosse menor que 79. Isto ocorre pela lei matemática conhecida como **Princípio da casa dos pombos**.
  Sabendo disso, você escolheu um tamanho de tabela maior que 79, mas menor que 676 já que isso seria desperdiçar muito espaço em sua mente. Ou seja, ou seja você escolheu uma tabela de tamanho $M = 100$. Porém, ainda viu que colisões continuam a ocorrer, mesmo tendo espaços suficientes para todas as chaves. O vilão desta vez é o **Paradoxo do aniversário**. Este paradoxo explica sobre a probabilidade de duas pessoas fazerem aniversário no mesmo dia em um grupo de n indivíduos escolhidos aleatoriamente. Por exemplo, de acordo com ele, são necessárias apenas 23 pessoas para que se obtenha 50% de probabilidade de que dois deles façam aniversário no mesmo dia. Essa probabilidade salta para 90% com 70 pessoas.
  O paradoxo do aniversário se aplica a colisões em tabelas hash da seguinte maneira: o número de pessoas que fazem aniversário corresponde ao número de chaves que inserimos na tabela hash. A capacidade da tabela substitui o número de dias em um ano, 365. Com isso em mente, o paradoxo do aniversário nos diz que há uma grande probabilidade de que duas de nossas 79 chaves colidam dada uma tabela pequena com 100 posições. A Wikipedia inclusive fornece uma fórmula que permite calcular essa probabilidade de colisão.

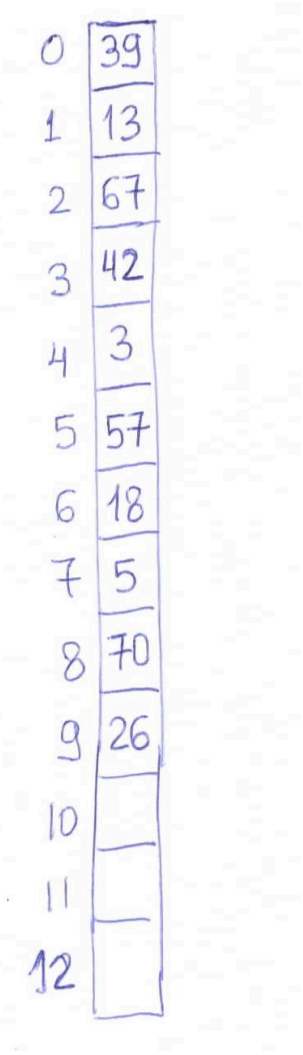
- **Questão 3.** Dada a entrada $\{42, 39, 57, 3, 18, 5, 67, 13, 70, 26\}$, e um tamanho de tabela fixo de $M = 13$, e uma função de hash $h(x) = x \bmod 13$, mostre o resultado da inserção das chaves acima em ordem para os seguintes tipos de tabela:

  1. Tabela hash com encadeamento exterior
     Solução: Neste caso, temos que as chaves podem ser reescritas da seguinte forma:
     $42 = 3 * 13 + 3$,
     $39 = 3 * 13 + 0$,
     $57 = 4 * 13 + 5$,
     $3 = 0 * 13 + 3$,
     $18 = 1 * 13 + 5$,
     $5 = 0 * 13 + 5$,
     $67 = 5 * 13 + 2$,
     $13 = 1 * 13 + 0$,
     $70 = 5 * 13 + 5$,
     $26 = 2 * 13 + 0$.
     Logo, a situação final da tabela fica:

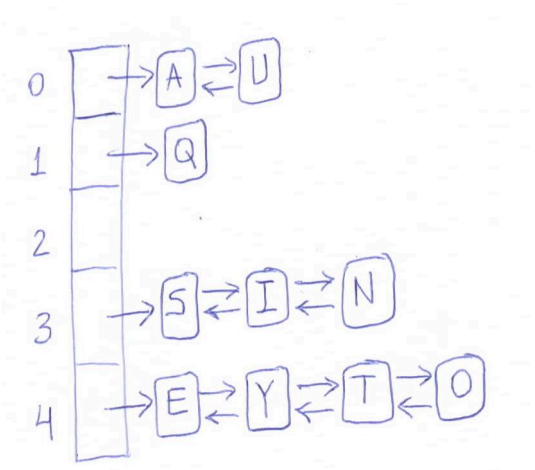
  2. Tabela hash de sondagem linear.
     Solução: Neste caso, listamos abaixo as chaves e quantas sondagens foram feitas para que cada uma delas chegasse ao seu bucket final:
     $42 = 3 * 13 + 3$, sondagens = 1
     $39 = 3 * 13 + 0$, sondagens = 1
     $57 = 4 * 13 + 5$, sondagens = 1
     $3 = 0 * 13 + 3$, sondagens = 2
     $18 = 1 * 13 + 5$, sondagens = 2

$5 = 0 * 13 + 5$, sondagens = 3
$67 = 5 * 13 + 2$, sondagens = 1
$13 = 1 * 13 + 0$, sondagens = 2
$70 = 5 * 13 + 5$, sondagens = 4
$26 = 2 * 13 + 0$, sondagens = 10.
Logo, a situação final da tabela fica:



- **Questão 4.** Forneça o conteúdo da tabela hash resultante quando você insere pares com as chaves E A S Y Q U T I O N nessa ordem em uma tabela hash inicialmente vazia de tamanho $M = 5$, usando encadeamento exterior com listas não ordenadas. Use a função de hash $f(k) = 11k \bmod M$ para transformar a k-ésima letra do alfabeto em um índice $f(k)$ tabela hash. **Atenção:** Como as chaves são caracteres, use o valor equivalente delas na tabela ASCII.

  Solução: O valor em decimal na tabela ASCII de cada chave é: E = 69, A = 65, S = 83, Y = 89, Q = 81, U = 85, T = 84, I = 73, O =79, N = 78.
  Tomando a função hash $f(k) = 11k \bmod 5$, cada chave é mapeada em um bucket da tabela como a seguir:
  $f(E) = 759 \bmod 5 = 4$
  $f(A) = 715 \bmod 5 = 0$
  $f(S) = 913 \bmod 5 = 3$
  $f(Y) = 979 \bmod 5 = 4$
  $f(Q) = 891 \bmod 5 = 1$
  $f(U) = 935 \bmod 5 = 0$
  $f(T) = 924 \bmod 5 = 4$
  $f(I) = 803 \bmod 5 = 3$
  $f(O) = 869 \bmod 5 = 4$
  $f(N) = 858 \bmod 5 = 3$
  Logo, a situação final da tabela fica:



◀ [visualgo] Tabela hash com encadeamento exterior

Seguir para...