Instituto de Computação – UNICAMP
MC202 – Estruturas de Dados – 2º Semestre de 2020
https://www.ic.unicamp.br/~lehilton/mc202ab/

Exercícios de fixação - **Tabela de espalhamento**

**Questão 1.** Suponha que queremos guardar os dados de uma turma de de alunos ingressantes em uma tabela de espalhamento. Escreva uma boa função de hashing para isso.

**Questão 2.** Quais as complicações para a remoção de tabelas de espalhamento que utilizam endereçamento aberto? Especifique um TAD "Tabela de Espalhamento" com endereçamento aberto e escreva as `structs` correspondentes.

**Questão 3.** Considere uma função de *hashing* $h(k)$ que mapeia uma string a um número entre 0 e 10. A função é calculada da seguinte forma: dada uma palavra, conte o número de caracteres e devolva o resto da divisão desse número por 10.

***(a)*** Desenhe a tabela de de espalhamento usando listas ligadas e endereçamento aberto com reespalhamento linear após a inserção das seguintes palavras: "De", "tudo", "ao", "meu", "amor", "serei", "atento".

***(b)*** Essa é um boa função de hashing? Justifique sua resposta descrevendo o que se espera de uma boa função de hashing e relacionando ao conjunto de chaves do domínio da função.

**Questão 4.** [GPT] Suponha que uma tabela de hashing vai armazenar chaves compostas por pares de inteiros $(x,y)$. Que estratégia poderia ser usada para computar uma função de hashing para o par que fosse independente da ordem dos elementos, isto é, $h(x,y) = h(y,x)$?

**Questão 5.** Como visto, normalmente uma boa função de hashing depende de todos os bits de sua chave. Isso pode ser entendido verificando o seguinte exemplo ruim de função: pegue a primeira letra de uma palavra e mapeie em um vetor com posições de 0 a 25. Um problema é que isso pode acarretar muitas colisões, já que as palavras não são divididas uniformemente; outro problema é que se o número de elementos for grande, então deveríamos gerar índices maiores que 25. Tendo isso em mente, explique porque a função de hashing $h(k) = k \bmod 2^i$, para algum $i$ inteiro não é uma boa função de hashing em geral.

**Questão 6.** Considere a seguinte afirmação: *"Tabelas de Espalhamento são estruturas de dados excelentes para uma gama enorme de aplicações. Seu tempo de execução é quase sempre $O(1)$, o que é muito melhor que o tempo médio de busca ou inserção de uma árvore balanceada $O(\log n)$. Não é por isso, no entanto, que sempre usaremos tabela de espalhamento para implementar um conjunto dinâmico."* Concorde ou discorde da frase. Pense em exemplos ou contraexemplos de aplicações que corroborem com sua posição. Escreva um parágrafo descrevendo o que é cada uma das estruturas juntamente com as operações e em que situações são adequadas.

*(vide verso)*

**Questão 7. Outras aplicações:** Funções de hashing podem ser aplicadas em outros contextos:

- *Verificação de paridade:* para evitar erros de transmissão, podemos, além de informar uma chave, transmitir o resultado da função de hashing. Exemplos:
  - dígitos verificadores
  - sequências de verificação para arquivos (*checksum* MD5 e SHA)
- *Segurança:* por definição, calcular o resultado da função de hashing para um objeto **deve** ser rápido; o problema inverso é o seguinte: dado um número $n$, queremos encontrar algum objeto $x$ tal que $f(x) = n$. Uma função de hashing cuja inversa seja "difícil" de calcular é considera segura.

***(a)*** O número de CPF é composto por 9 dígitos e mais 2 dígitos verificadores. Os dígitos verificadores nesse caso servem para evitar erros de digitação. Sabemos que a chance de uma pessoa digitar errado um dígito de seu CPF é muito maior do que a probabilidade de errar dois dígitos, que é maior que de errar 3 e assim por diante. Considerando isso, elenque boas propriedades da função de hashing que gera dígitos para o CPF.

***(b)*** Os números de CPF são distribuídos pela Receita Federal de acordo com a localidade. Assim, o nono dígito sempre é o mesmo para as pessoas que emitiram o CPF em uma cidade. Concorde ou discorde: em uma função de hashing, sempre devemos verificar todos os bits da entrada. Justifique.

***(c)*** Tente explicar o porquê da definição da função de hashing segura.

***(d)*** (extra) Uma função de hashing que usa o método da divisão $h(x) = x \bmod p$ não é segura. Para ver isso, basta notar que, se $y = x \bmod p$, então $0 \leq x < p$ e, portanto $y$ já inverte a função de hashing, isso é, $f(y) = y$. Argumente que o método da multiplicação também não fornece uma função de hashing segura.