Aula 21
Tabelas de Dispersão

Tabelas de endereçamento aberto c/ dupla dispersão

Algoritmos e Estruturas de Dados

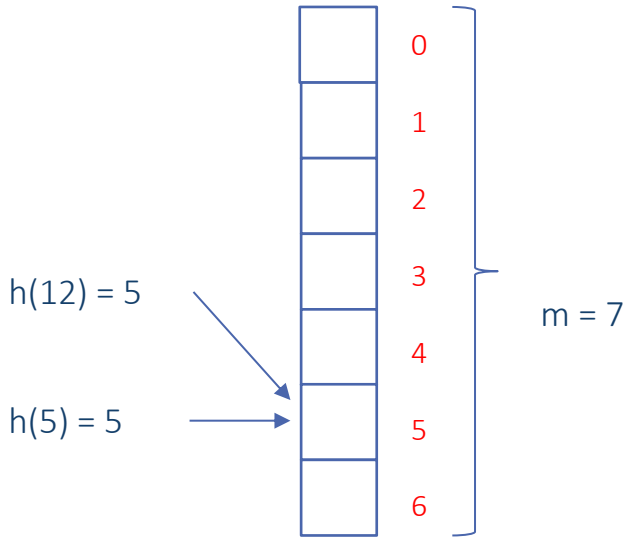
Tratamento de colisões

- O que fazer quando duas chaves diferentes têm o mesmo valor de hash?
- Tratamento de colisões
  - Várias soluções
  - Tabelas de encadeamento separado

    Separate chaining
  - Tabelas de endereçamento aberto

    Linear probing

    Double hashing



Tabelas de endereçamento aberto

Com dupla dispersão

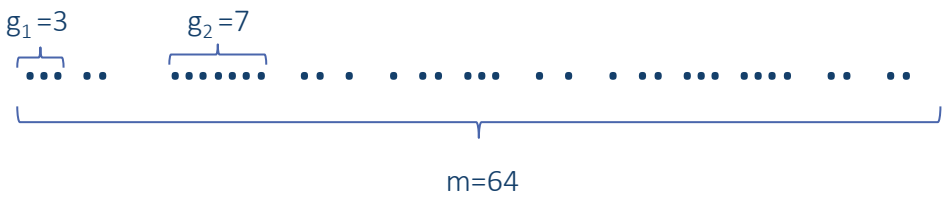
Double hashing

Problema com exploração linear

- Exploração linear causa problemas de aglomeração

Aglomeração

- Tempo de pesquisa/inserção depende do tamanho dos grupos existentes na tabela de dispersão
  - Ex: Inserção no grupo 1 necessita de 3 comparações



  - Grupos longos poderão ser comuns
  - Grupo longo tem uma maior probabilidade de aumentar do que grupos pequenos

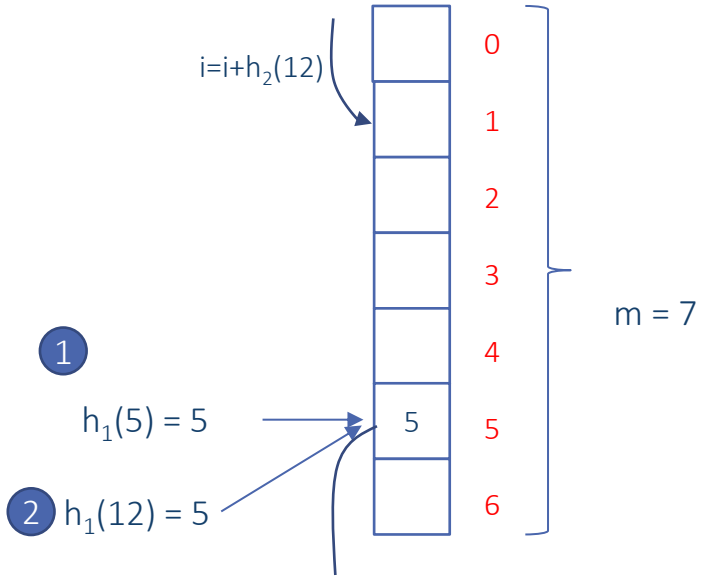
    Ex:

    probabilidade da próxima chave ser colocada no grupo 2 = 7/64

    probabilidade da próxima chave ser colocada no grupo 1 = 3/64

Tabela de endereçamento aberto c/ dupla dispersão

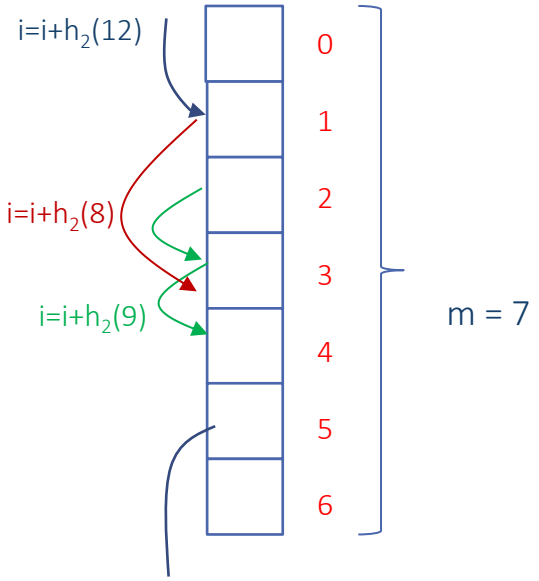
- A técnica de dupla dispersão permite lidar com o problema de aglomeração
  - Em vez de percorrer o array de forma linear
  - Usar uma segunda função de *hash* h_2



Dupla dispersão (double hashing)
Quando existe colisão, esta técnica vai explorar as próximas posições usando uma segunda função de dispersão
$i=(i+h_2(k)) \bmod m$

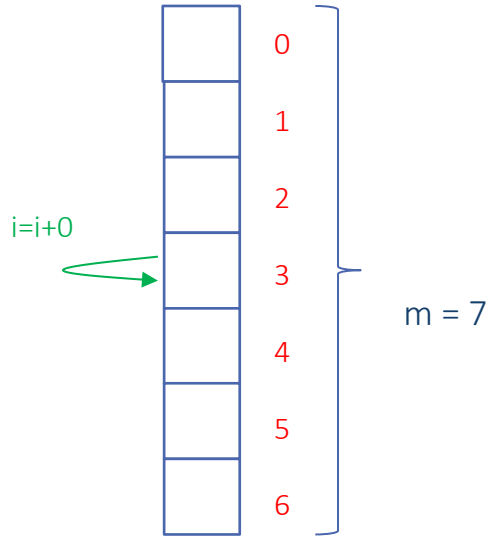
Tabela de endereçamento aberto c/ dupla dispersão

- Pq uma função de dispersão h_2?
  - Chaves diferentes vão dar origem a padrões de exploração diferentes
  - Vai evitar a formação de agrupamentos



Propriedades função dispersão h_2

- Função de dispersão $h_2(k)$ deve
  - Ser muito eficiente
  - Idealmente $h_2(k) \neq h_1(k)$
  - $h_2(k)$ *nunca pode ser 0 ou m*



Funções de dispersão h_1 e h_2

- *Funções de dispersão muito usadas*
  - $h_1(k) = hashcode(k) \% m$
- Dois exemplos de funções h_2 muito usadas
  - $h_2(k) = p - (hashcode(k) \% p)$

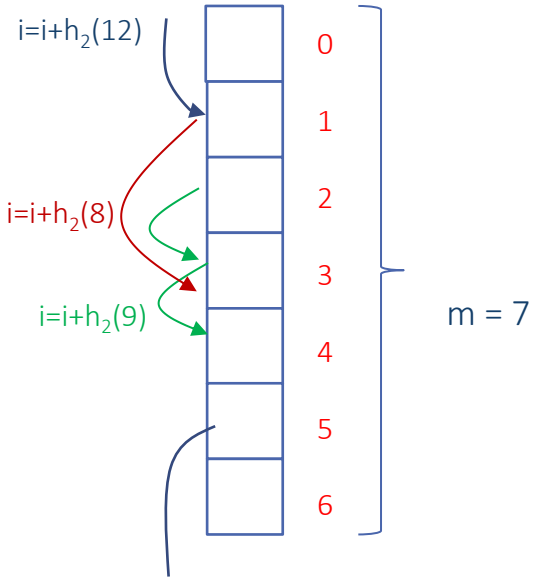
    sendo p um número primo < m
  - $h_2(k) = 1 + (hashcode(k) \% p)$

    sendo p um número primo < m

Observação:
c/dupla dispersão
$h_2(k)$ nunca pode ser 0 ou m

Função de dispersão h_1 e h_2

- *Exemplo*
  - *m = 7, p = 5*
  - *h1(12) = 12%7 = 5*
  - *h2(12) = 5 – (12%5) = 5-2=3*
  - *h1(8) = 8%7 = 1*
  - *h2(8) = 5 – (8%5) = 5-3=2*
  - *h1(9) = 9%7 = 2*
  - *h2(9) = 5 – (9%5) = 5-4=1*



Problema com dupla dispersão

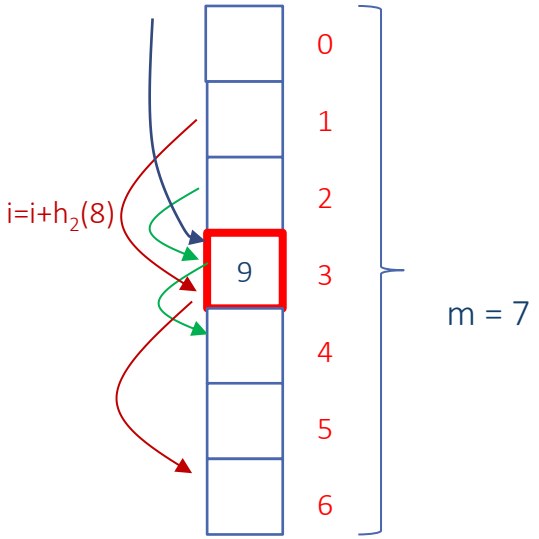
- A dupla dispersão tem a vantagem de evitar agrupamentos
- Mas torna a remoção de chaves bastante mais complexa
  - Remoção usada na pesquisa linear não funciona
  - Quando removemos uma chave (ex 9)

    Existem várias sequências de exploração que passam pela chave na posição 3

    A chave da posição 3 pode fazer parte de uma ou mais dessas sequências

    E não sabemos quais!!!

    Ao removermos a chave da posição 3 corremos o risco de quebrar essas sequências



Remoção em dupla dispersão

- Numa tabela com dupla dispersão usa-se remoção preguiçosa
- Lazy delete
  - Em vez de apagarmos realmente uma chave da tabela
  - Marcamos essa chave como “apagada” mas não a removemos

    Várias formas de o fazer

    a) Usar uma variável para indicar o estado (vazia, ocupada, apagada)

    b) Ou usar uma marca especial para indicar uma chave apagada

    Ex. Podemos considerar uma chave com um valor *null como uma chave apagada*

Remoção preguiçosa

- Operações de pesquisa e remoção tratam posições de chaves "apagadas" como se estivessem ocupadas

- Operação de inserção
  - Se durante a exploração de uma operação de inserção encontramos a chave que estamos a tentar inserir

    Mas entretanto essa chave era uma chave "apagada"

    Podemos inserir o valor na chave "apagada"

    A chave deixou de estar "apagada"

Remoção preguiçosa

- Problema com chaves apagadas, mas não removidas

- Quando existem muitas, tornam a pesquisa mais ineficiente
  - Podemos ter que iterar sobre várias chaves apagadas até encontrar a chave pretendida

Resolução de chaves apagadas

- Chaves apagadas podem ser explicitamente resolvidas nas seguintes situações
  - Quando se insere um novo valor para a chave (que tinha sido apagada anteriormente)

    A chave deixa de ser considerada apagada, passa a ter um valor

  - Quando se redimensiona a tabela de dispersão

    As chaves "apagados" não são colocadas na nova tabela

  - Quando o número de chaves apagadas é muito grande

    Ex, apagadas > 20% tamanho total da tabela de dispersão

    Fazemos uma nova tabela com o mesmo tamanho

    Colocamos novamente as chaves não apagados

    Ignoramos as chaves "apagadas"

Complexidade Temporal

- Número médio de comparações para uma tabela de dupla dispersão com factor de carga α
  - $\sim \frac{1}{\alpha} ln \frac{1}{1-\alpha}$ no caso de search hit
  - $\sim \frac{1}{1-\alpha}$ no caso de search miss/insert
- Para α=50% (=) α=1/2
  - ~ 2 ln 2 = 1.38 no caso de search hit
  - ~ 2 no caso de search miss/insert

Complexidade Temporal

	Caso Médio		Pior Caso	
	Inserção	Pesquisa	Inserção	Pesquisa
Árvore Red-Black	$\log_2 n$	$\log_2 n$	$2 \log_2 n$	$2 \log_2 n$
Tabela de encadeamentos separados	4* 2**	2.5* 1.5**	4-8* 2-4**	4-8* 2-4**
Tabela de endereçamento aberto c/ exploração linear	2.5***	1.5***	2-4**	2.5**
Tabela de endereçamento aberto c/ dupla dispersão	**2*** **	**1.38*** **	**n 2-4*** **	**n 2-4*** **

Pior caso se a função de hash não for boa

Observação: dupla dispersão consegue uma eficiência semelhante a uma **TES com m = n/2**, mas **gastando significativamente menos memória**

* assumindo m = n/4
** assumindo m = n/2
***assumindo um factor de carga de 50%