Algoritmos e Estrutura de Dados

Aula 10 – Estrutura de Dados: Tabelas Hash

Prof. Tiago A. E. Ferreira

Introdução

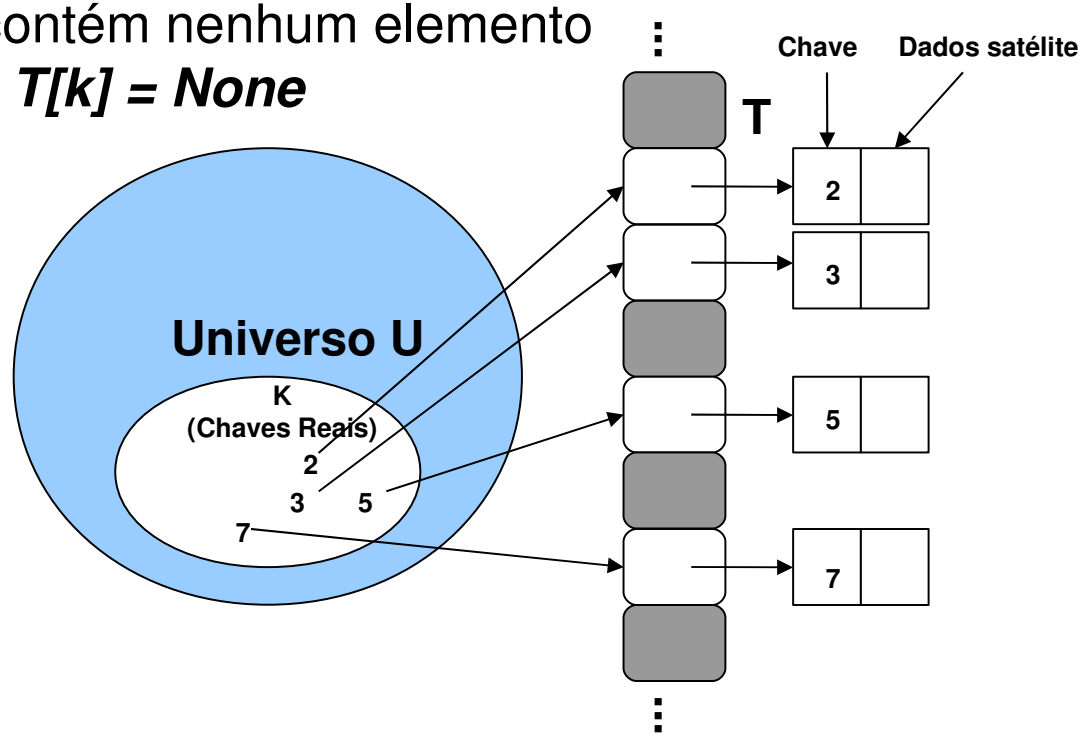
- Muitas aplicações exigem de forma dinâmica apenas as operações de dicionário:
  - Inserir (*insert)*
  - Pesquisar (*search*)
  - Deletar (*delete*)
- Uma **tabela hash** é uma estrutura de dados eficiente para implementar dicionários!

Tabelas Hash de Endereço Direto

- Dado um universo de chaves U
  - Se a cardinalidade de U é pequena, o endereçamento direto é uma técnica simples que funciona bem!
- Suponha que uma dada aplicação necessite de um conjunto dinâmico onde cada elemento tenha uma chave definida a partir do universo
  - **U = {0, 1, ..., m-1}**
  - Onde **m** não é muito grande, e não há dois elementos com a mesma chave
- Para tanto vamos utilizar uma **tabela de endereçamento direto T[0..m-1]**

Tabelas de Endereço Direto

- Cada **slot**, ou posição, de T corresponde a uma chave no universo **U**
  - A posição **k** aponta para um elemento no conjunto com chave **k**.
    - Se o conjunto não contém nenhum elemento com chave **k**, então ***T[k] = None***



Tabelas de Endereço Direto

- Para estas tabelas hash também é possível armazenar o dado satélite na própria tabela T
  - Isso irá economizar memória, visto que a chave pode ser encarada como o índice da tabela T.
  - Contudo, se feito isto, algo deve ser realizado para saber se a posição está vazia

Operações de um Dicionário

- DIRECT-ADDRESS-SEARCH(T,k)

  return T[k]

- DIRECT-ADDRESS-INSERT(T,k)

  T[chave[k]] ⟵ x

- DIRECT-ADDRESS-DELETE(T,k)

  T[chave[k]] ⟵ None

- Observe que todas estas operações são rápidas: é necessário apenas tempo **O(1)**

Tabelas Hash

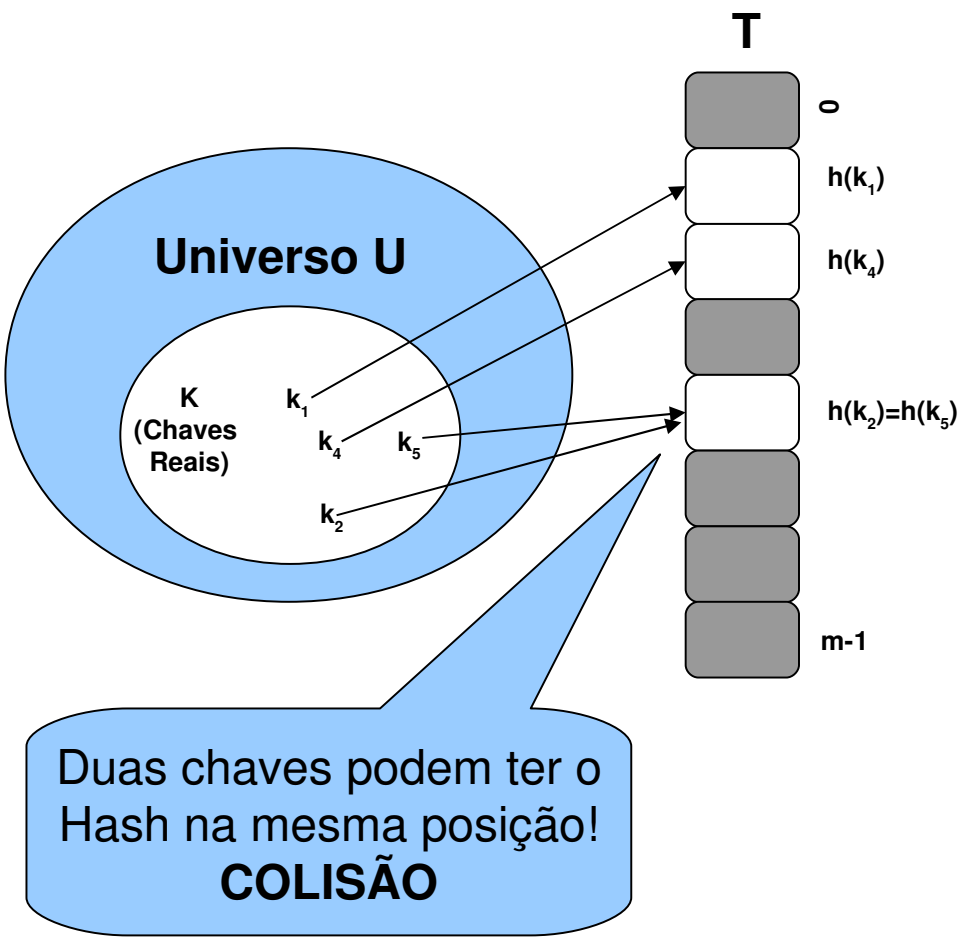
- Qual a dificuldade do endereçamento direto?
  - Se o universo **U** é muito grande, o armazenamento de uma tabela **T** de tamanho **|U|** pode ser impraticável!
  - Além do que a quantidade de chaves reais utilizadas pode ser muito menor que **|U|** implicando que a maior parte de T seria desperdiçada.
    - Para este caso, uma tabela hash exige muito menos espaço de endereçamento do que uma tabele de endereçamento direto!
    - Os requisitos de armazenamento são Θ(|k|)

Tabela Hash

- Em uma tabela hash, um elemento com chave k é armazenado na posição *h(k)*
  - Ou seja, ***função hash h*****,** que é utilizada para calcular a posição do elemento a partir da chave **k**.
- Função Hash
  - Mapeamento do Universo **U** de chaves nas posições da tabela hash **T[0..m-1]**

$$h : U \rightarrow \{0,1, \ldots, m-1\}$$

Tabelas Hash



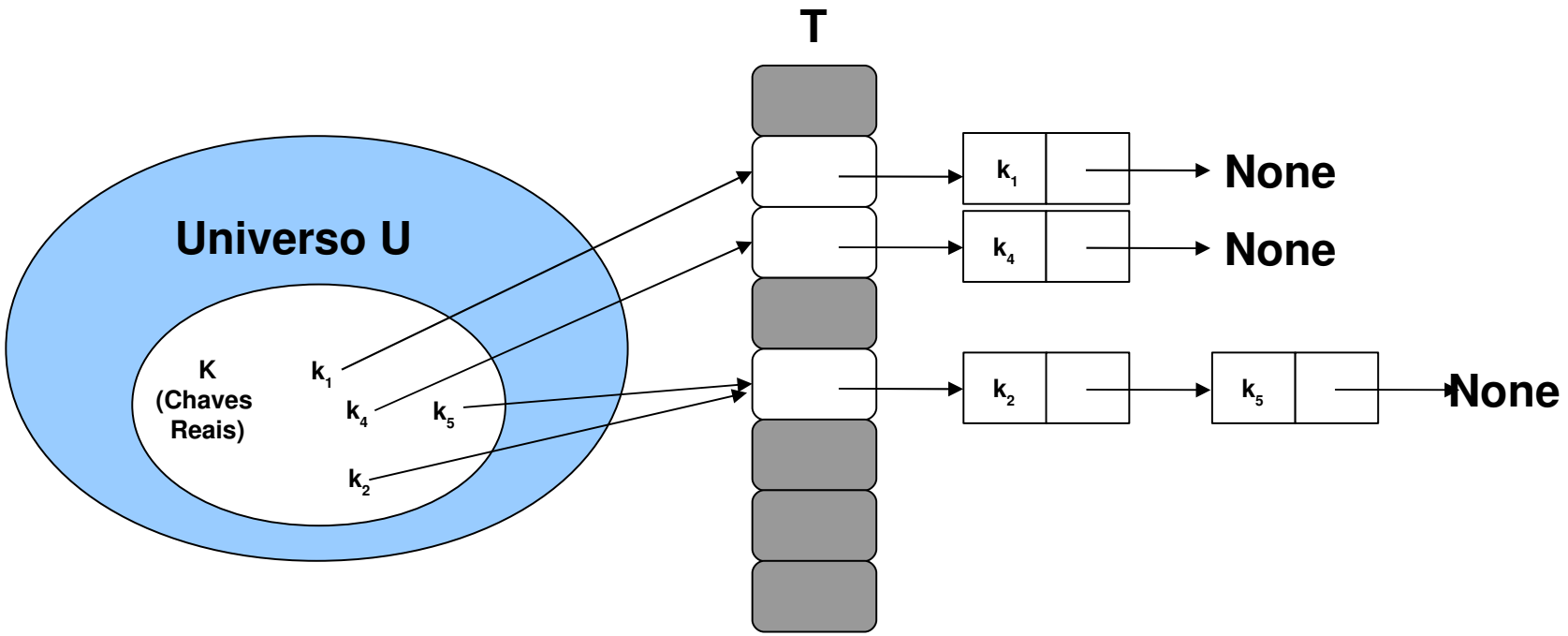
- É dito que um elemento com chave **k** *efetua o hash* para a posição *h(k)*
- É dito também que *h(k)* é o **valor hash** da chave *k*.

Tabelas Hash - Colisões

- A melhor solução é evitar as colisões!
  - Contudo, |**U**| > **m** avendo uma chance muito grande para haver colisões!
  - Assim, temos que desenvolver técnicas para poder tratar as colisões.

Resolução de Colisões por Encadeamento

- Uma possível solução é colocar todos os elementos que efetuam o hash para a mesma posição como uma lista ligada



Resolução de Colisões por Encadeamento

- As operações de dicionário quando as colisões são resolvidas por encadeamento:
  - CHAINED-HASH-INSERT(T,k)
    - Insere **x** no início da lista ***T[h(chave[x])]***
  - CHAINED-HASH-SEARH(T,k)
    - Procura por um elemento com a chave **k** na lista ***T[h(k)]***
  - CHAINED-HASH-DELETE(T,k)
    - Elimina o elemento **x** da lista ***T[h(chave[x])]***

Análise do Hash com Encadeamento

- Qual a qualidade de execução do hash com encadeamento?
  - Em particular, quanto tempo ele leva para procura um elemento com uma determinada chave?
- Para uma tabela hash com **m** posições e **n** elementos:
  - **Fator de Carga α:** n/m
    - Todas as análises serão feitas em termos de **α!**

Análise do Hash com Encadeamento

- **Pior Caso:**
  - Todas as **n** chaves executam o hash na mesma posição!
    - É criada uma lista de comprimento **n**.
  - O tempo para a pesquisa é **n** mais o tempo de calcular a função hash.
    - Esta caso não é melhor que utilizar uma lista ligada para representar todos os elementos

Análise do Hash com Encadeamento

- **Caso Médio:**
  - Irá depender com a função hash irá distribuir o conjunto de chaves entre as **m** posições.
  - Vamos supor que um elemento dado tem igual probabilidade de efetuar o hash para qualquer das **m** posições.
    - **Hipótese de hash uniforme.**
    - Denota-se o comprimento da lista **T[j]** por $\mathbf{n}_j$, onde,

$$n = n_0 + n_1 + \cdots + n_{n-1}$$

    - e,

$$\overline{n}_j = E[n_j] = \frac{n}{m} \rightarrow Fator\ \ de\ Carga$$

Análise do Hash com Encadeamento

- Supondo que o valor hash **h(k)** pode ser calculado em tempo **O(1)**
  - Desta forma, o tempo necessário para procurar um elemento **k** depende linearmente do comprimento das listas
  - Assim, é possível considerar duas situações:
    - Uma pesquisa não é bem-sucedida;
    - Uma pesquisa é bem-sucedida

Análise do Hash com Encadeamento

- Teorema 11.1
  - Em uma tabela hash na qual as colisões são resolvidas por encadeamento, uma pesquisa malsucedida demora o tempo esperado $\Theta(1+\alpha)$, sob a hipótese de hash uniforma simples.
- Teorema 11.2
  - Em uma tabela hash na qual as colisões são resolvidas por encadeamento, uma pesquisa bem-sucedida demora o tempo $\Theta(1+\alpha)$, na média, sob a hipótese de hash uniforme simples.

Análise do Hash com Encadeamento

- O que poemos concluir?
  - Se o número de posições na tabela hash é no mínimo proporcional ao número de elementos na tabela:
    - **n = O(m), e consequentemente, α = n/m = O(m)/m = O(1)**
    - **Portanto, a pesquisa demora um tempo constante em média**
    - **Dado que a inserção demora O(1) e a eliminação demora O(1) para listas duplamente ligadas!**
    - **Todas as operações de dicionário podem ser admitidas em tempo O(1) em média!**

Exercícios Práticos

- Exercício 1: Provar os teoremas 11.1 e 11.2

- Exercício 2: Implemente em Python uma tabela hash com colisões resolvidas por encadeamento. Seja a tabela com 9 osições, e seja a funções de hash h(k)=k mod 9. Demostre a inserção das chaves 5, 28, 19, 15, 20, 33, 12, 17 e 10.