Algoritmos e Estruturas de Dados III

3º Período Engenharia da Computação

Prof. Edwaldo Soares Rodrigues
Email: edwaldoroadsf1@yahoo.com.br

UNIVERSIDADE
DO ESTADO DE MINAS GERAIS | UEMG
UNIDADE DIVINÓPOLIS

Transformação de Chave (*Hashing*)

- As estruturas baseadas em transformação de chave utilizam um arranjo para guardar os elementos do conjunto;

- Quando uma chave é pesquisada, esta chave é transformada em um número por meio de uma operação de transformação de chave;

- O número retornado representa uma posição em um arranjo, onde o elemento é guardado;

Transformação de Chave (*Hashing*)

- Os registros armazenados em uma tabela são diretamente endereçados a partir de uma transformação aritmética sobre a chave de pesquisa.

- *Hash* significa:
  - Fazer picadinho de carne e vegetais para cozinhar.
  - Fazer uma bagunça. (Webster's New World Dictionary)
  - Espalhar x Transformar (informática x computação)

Transformação de Chave (*Hashing*)

- Um método de pesquisa com o uso da transformação de chave é constituído de duas etapas principais:
  - 1 - Computar o valor da **função de transformação**, a qual transforma a chave de pesquisa em um endereço da tabela.
  - 2 - Considerando que duas ou mais chaves podem ser transformadas em um mesmo endereço de tabela, é necessário existir um método para lidar com **colisões**.
- Qualquer que seja a função de transformação, algumas colisões irão ocorrer fatalmente, e tais colisões têm de ser resolvidas de alguma forma.
- Mesmo que se obtenha uma função de transformação que distribua os registros de forma uniforme entre as entradas da tabela, existe uma alta probabilidade de haver colisões.

Transformação de Chave (*Hashing*)

- O paradoxo do aniversário (Feller,1968, p. 33), diz que em um grupo de 23 ou mais pessoas, juntas ao acaso, existe uma chance maior do que 50% de que 2 pessoas comemorem aniversário no mesmo dia.
- Assim, se for utilizada uma função de transformação uniforme que enderece 23 chaves randômicas em uma tabela de tamanho 365, a probabilidade de que haja colisões é maior do que 50%.

Transformação de Chave (*Hashing*)

- Alguns valores de p para diferentes valores de N, onde M = 365.

N	P
10	0,883
22	0,524
23	0,493
30	0,303

- Para N pequeno a probabilidade p pode ser aproximada por $p \approx N(N-1))/730$. Por exemplo, para N = 10 então p ≈ 88%.

Funções de Transformação

- Uma função de transformação deve mapear chaves em inteiros dentro do intervalo [0...M − 1], onde M é o tamanho da tabela.

- A função de transformação ideal é aquela que:
  - Seja simples de ser computada.
  - Para cada chave de entrada, qualquer uma das saídas possíveis é igualmente provável de ocorrer.

Método mais Usado

- Usa o resto da divisão por M .

$$h(K) = K \% M \quad \text{(em linguagem C)}$$

onde K é um inteiro correspondente à chave.

Método mais Usado

- Cuidado na escolha do valor de M. M deve ser um número primo, mas não qualquer primo: devem ser evitados os números primos obtidos a partir de

$$b*i \pm j$$

onde b é a base do conjunto de caracteres (geralmente b = 64 para BCD, 128 para ASCII, 256 para EBCDIC, ou 100 para alguns códigos decimais), e i e j são pequenos inteiros.

Transformações de Chaves não numéricas

- As chaves não numéricas devem ser transformadas em números:

$$K = \sum_{i=1}^{n} \text{Chave}[i] \times p[i],$$

- n é o número de caracteres da chave.
- Chave[i] corresponde à representação ASCII do i-ésimo caractere da chave.
- p[i] é um inteiro de um conjunto de pesos gerados randomicamente para $1 \leq i \leq n$.

Transformações de Chaves não numéricas

- Vantagem de se usar pesos: Dois conjuntos diferentes de pesos p1[i] e p2[i], $1 \leq i \leq n$, leva a duas funções de transformação h1(K) e h2(K) diferentes.

Transformação de Chave (*Hashing*)

- Uma possibilidade para efetuar o armazenamento de informações usando Hashing consiste no uso de um arranjo, que analogamente pode ser comparado a uma tabela, desta forma o termo Tabela Hash é muito utilizado na computação, referindo-se a transformação de chave (hashing);

Transformação de Chave (*Hashing*)

- O tamanho inicial desta tabela depende da implementação. Mas suponha que a tabela Hash tenha espaço alocado para 7 elementos como apresentado na figura a seguir:

Transformação de Chave (*Hashing*)

- Suponha agora que queremos inserir na tabela o elemento 32. A função de transformação h(x) deverá transformar esta chave 32 em uma posição do arranjo

Transformação de Chave (*Hashing*)

- A função de transformação h(x) mais comum para números inteiros é h(x) = x % m, onde m é o tamanho da tabela hash e x é o valor da chave
- 13 % 3 = 1
- 3 % 13 = 3

$$h(32) = 32 \% 7 = 4$$

Transformação de Chave (*Hashing*)

- Um problema óbvio com tabelas hash é que a transformação de chave pode querer colocar um outro elemento na mesma posição da tabela

$$h(46) = 46 \% 7 = 4$$

Transformação de Chave (*Hashing*)

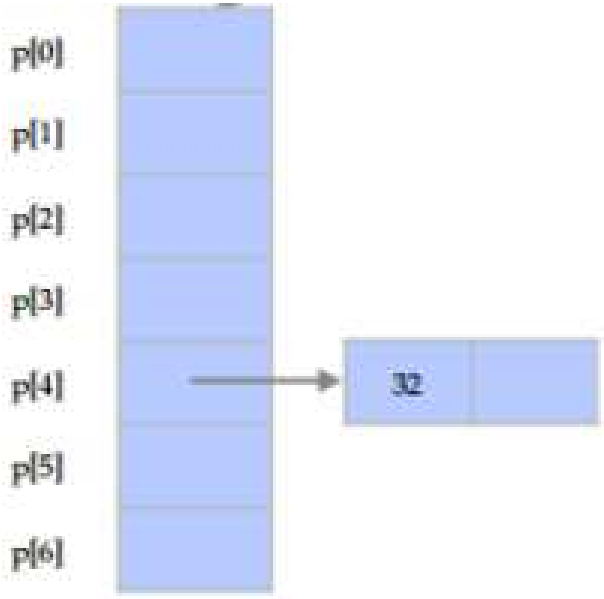
- Tratamento de colisões:
  - Listas encadeadas;
  - Endereçamento aberto;

Transformação de Chave (*Hashing*)

- Tratamento de colisões:
  - Listas encadeadas:
    - A estratégia mais comum para tratamento de colisões é transformar este arranjo de elementos em um arranjo com ponteiros para listas encadeadas de elementos;

Transformação de Chave (*Hashing*)

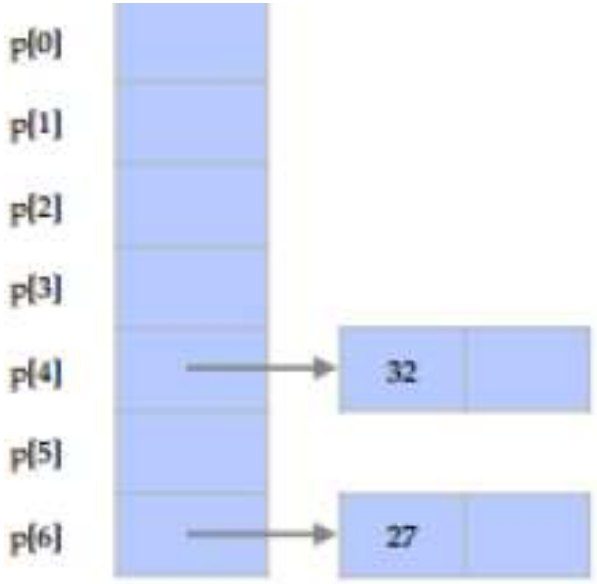
- Tratamento de colisões:
  - Listas encadeadas:
    - Por exemplo, ao se inserir o elemento 32, ele vai para a posição h(32) = 4 da tabela hash como um elemento em uma célula de uma lista encadeada;



$$h(32) = 32 \% 7 = 4$$

Transformação de Chave (*Hashing*)

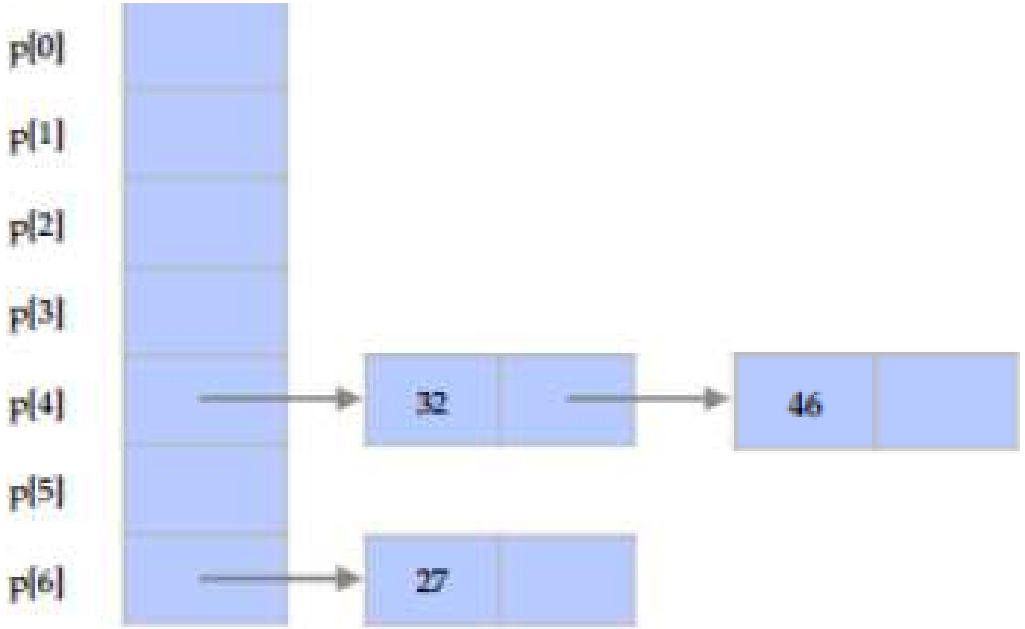
- Tratamento de colisões:
  - Listas encadeadas:
    - Cada elemento fica em uma célula que é composta de um elemento e um ponteiro para uma possível próxima célula;



$$h(27) = 27 \% 7 = 6$$

Transformação de Chave (*Hashing*)

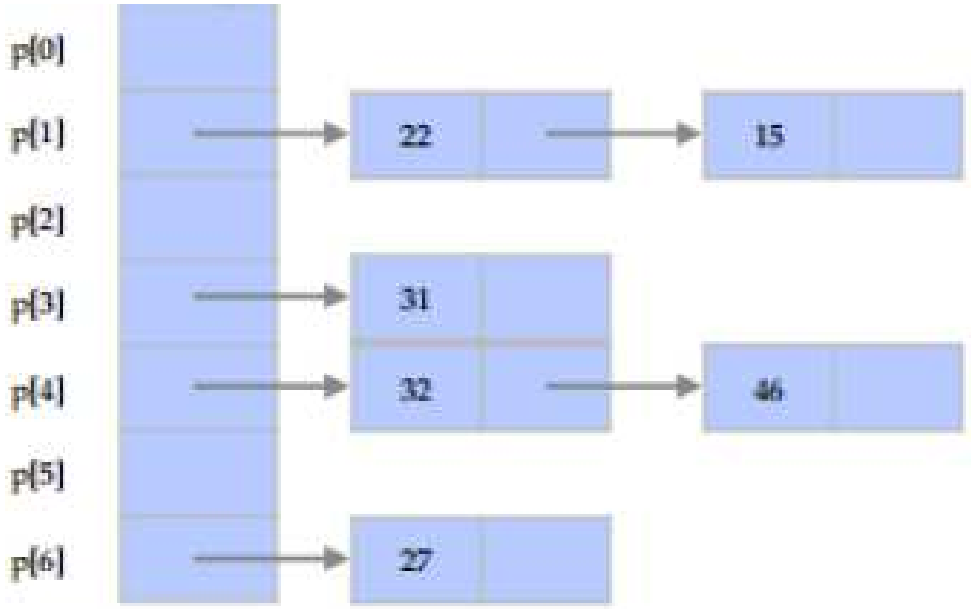
- Tratamento de colisões:
  - Listas encadeadas:
    - Em caso de colisão, o elemento é simplesmente colocado como último da lista;



$h(46) = 46 \% 7 = 4$

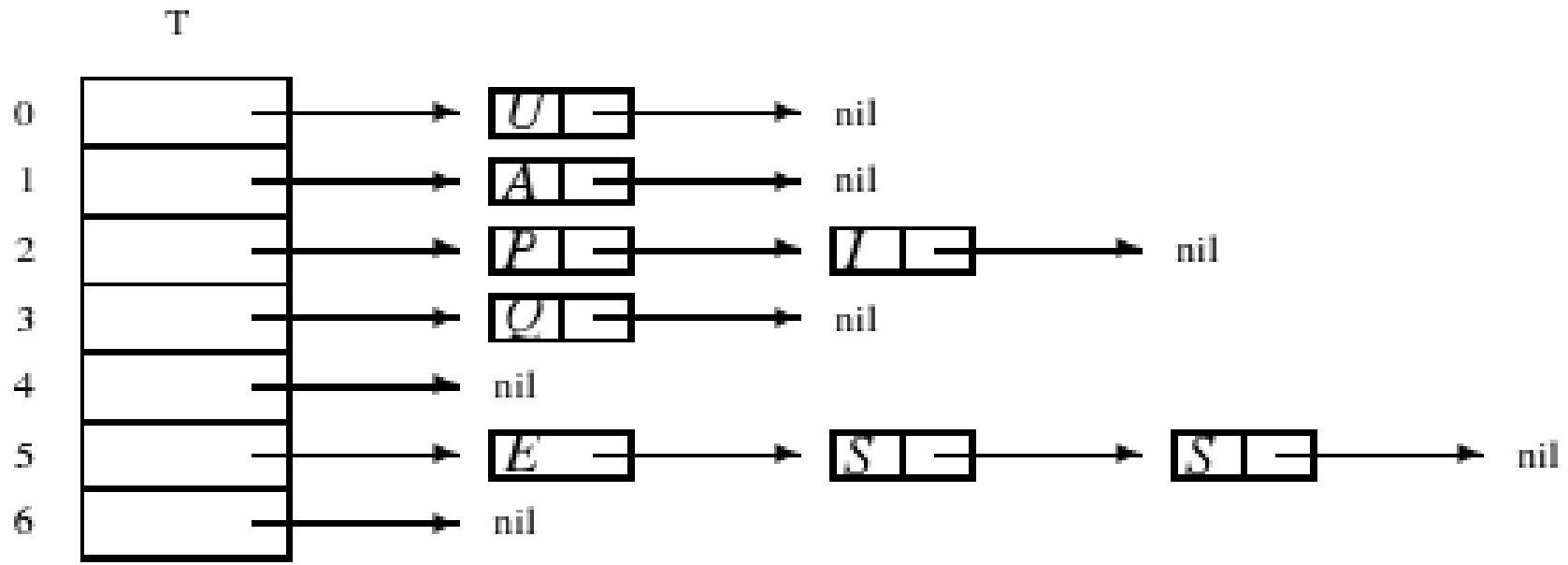
Transformação de Chave (*Hashing*)

- Tratamento de colisões:
  - Listas encadeadas:
    - Em caso de colisão, o elemento é simplesmente colocado como último da lista;



Transformação de Chave (*Hashing*)

- Tratamento de colisões:
  - Listas encadeadas:
    - Se a i-ésima letra do alfabeto é representada pelo número i e a função de transformação h(chave) = chave mod M é utilizada para M=7, o resultado da inserção das chaves PESQUISA na tabela é o seguinte:
    - Por exemplo: h(A) = h(1) = 1, h(E) = h(5) = 5, h(S) = h(19) = 5, entro outros;



Transformação de Chave (*Hashing*)

- Tratamento de colisões:
  - Listas encadeadas:
    - FERIADO;

Transformação de Chave (*Hashing*)

- Tratamento de colisões:
  - Endereçamento aberto:
    - Quando o número de registros a serem armazenados na tabela puder ser previamente estimado, então não haverá necessidade de usar apontadores para armazenar os registros;
    - Existem vários métodos para armazenar N registros em uma tabela de tamanho M > N, os quais utilizam os lugares vazios na própria tabela para resolver as colisões. (Knuth, 1973, p.518);

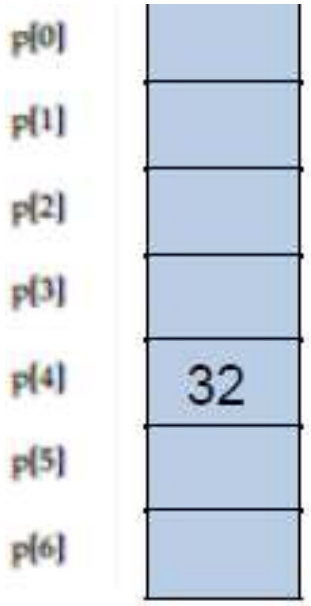
Transformação de Chave (*Hashing*)

- Tratamento de colisões:
  - Endereçamento aberto:
    - No endereçamento aberto todas as chaves são armazenadas na própria tabela, sem o uso de apontadores explícitos;
    - Existem várias propostas para a escolha de localizações alternativas. A mais simples é chamada de hashing linear, onde a posição h_j na tabela é dada por:

      $h_j = (h(x) + j) \bmod M$, para $1 <= j <= M-1$;

Transformação de Chave (*Hashing*)

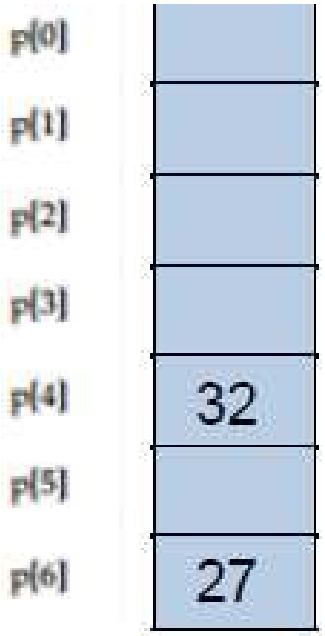
- Tratamento de colisões:
  - Endereçamento aberto:



$h(32) = 32 \% 7 = 4$

Transformação de Chave (*Hashing*)

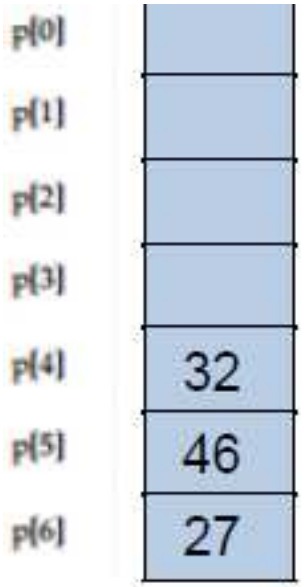
- Tratamento de colisões:
  - Endereçamento aberto:



$h(27) = 27 \% 7 = 6$

Transformação de Chave (*Hashing*)

- Tratamento de colisões:
  - Endereçamento aberto:



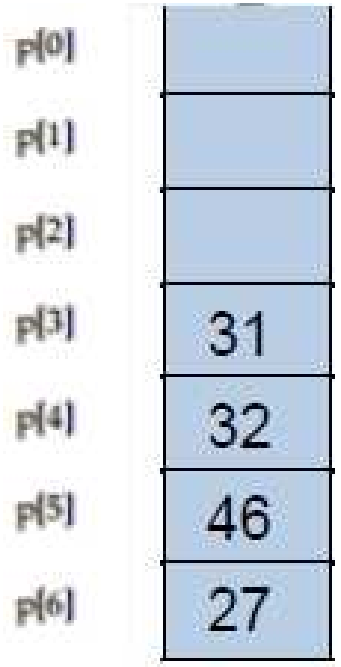
$h(46) = 46 \% 7 = 4$

Posição 4 já sendo utilizada, ou seja, houve **COLISÃO**!

Como ouve colisão, o novo elemento é armazenado na próxima posição vazia da Tabela Hash

Transformação de Chave (*Hashing*)

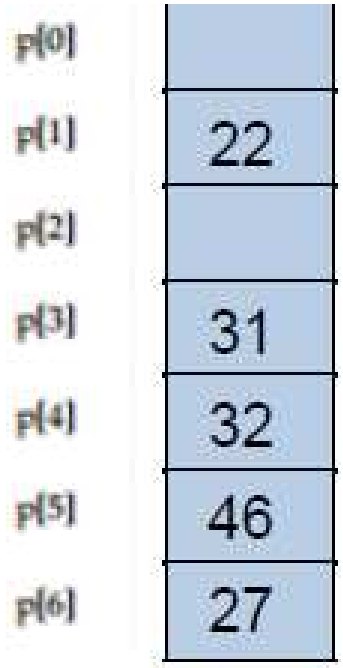
- Tratamento de colisões:
  - Endereçamento aberto:



h(31) = 31 % 7 = 3

Transformação de Chave (*Hashing*)

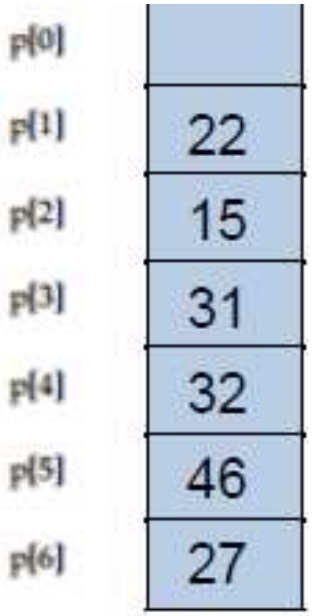
- Tratamento de colisões:
  - Endereçamento aberto:



h(22) = 22 % 7 = 1

Transformação de Chave (*Hashing*)

- Tratamento de colisões:
  - Endereçamento aberto:



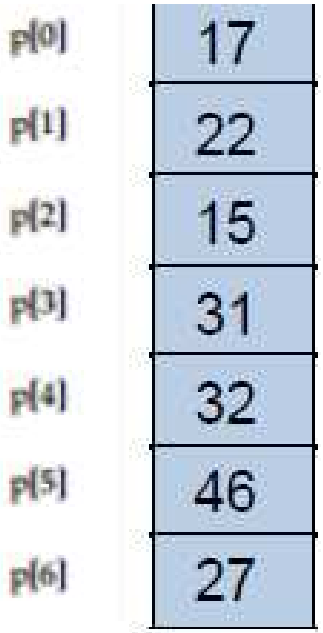
h(15) = 15 % 7 = 1

Posição 1 já sendo utilizada, ou seja, houve **COLISÃO**!

Como ouve colisão, o novo elemento é armazenado na próxima posição vazia da Tabela Hash

Transformação de Chave (*Hashing*)

- Tratamento de colisões:
  - Endereçamento aberto:



h(17) = 17 % 7 = 3

Posição 3 já sendo utilizada, ou seja, houve **COLISÃO**!

Como ouve colisão, o novo elemento é armazenado na próxima posição vazia da Tabela Hash

Transformação de Chave (*Hashing*)

- Tratamento de colisões:
  - Endereçamento aberto:
    - Se a í-ésima letra do alfabeto é representada pelo número i e a função transformação, onde M = 7;

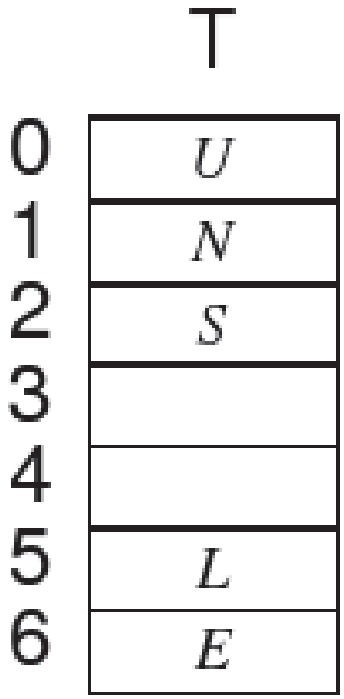
      h(chave) = chave mod M;

    - Então o resultado da inserção das chaves LUNES na tabela T usando hashing linear para resolver colisões é mostrado a seguir;

Exemplo

- Por exemplo:

h(L) = h(12) = 5, h(U) = h(21) = 0,
h(N) = h(14) = 0, h(E) = h(5) = 5,
h(S) = h(19) = 5.



Hashing - Análise

- A maior vantagem de tabelas hash é a sua eficiência em relação a custo médio para pesquisa, inserção e remoção;
- Para uma tabela onde os dados estão bem distribuídos o custo de qualquer operação na tabela é O(1);
- O STL já inclui estratégias para que os dados fiquem sempre bem distribuídos na tabela;

Hashing - Análise

- Para a estratégia de tratamento de colisão apresentada, no pior caso, se todos os elementos forem para a mesma posição, o custo de cada operação seria O(n);

- Porém, além da probabilidade deste caso ser desprezível, um tratamento de colisão por meio de árvores em vez de listas pode levar este pior caso a O(log n);

Hashing - Análise

- Para a estratégia de tratamento de colisão apresentada, no pior caso, se todos os elementos forem para a mesma posição, o custo de cada operação seria O(n);

- Porém, além da probabilidade deste caso ser desprezível, um tratamento de colisão por meio de árvores em vez de listas pode levar este pior caso a O(log n);

Hashing - Análise

- Mais ainda, é possível garantir de várias formas uma boa distribuição dos dados na tabela;
- Todas as tabelas hash alocam espaço para tabelas maiores quando o número de elementos cresce;
- Usualmente uma nova tabela maior é criada quando o número de elementos é maior que 70% do tamanho da tabela;
- Esta proporção é chamada de fator de carga;

Hashing - Análise

- Quando o tamanho da tabela aumenta uma operação O(n) copia os elementos para a nova tabela;
- Para evitar isto, é comum manter duas tabelas;
- Sempre que um elemento novo é inserido na tabela nova, um elemento da tabela antiga é transferido para a tabela nova com custo O(1);
- Porém, nesta estratégia, a busca de um elemento, apesar de ser ainda O(1), sempre precisará ser feita em duas tabelas;

Hashing - Análise

- A maior desvantagem das tabelas hash é que os dados ficam desordenados na tabela;
- Para retornar todos os itens em ordem é necessário colocar tudo para uma outra estrutura ordenada ou ordená-los em uma estrutura de sequência;
- Qualquer opção terá um custo O(n log n);
- Por isto, esta é uma estrutura a ser utilizada apenas quando os dados em ordem realmente não são necessários;

Hashing

- Exercício:
  - Faça o passo a passo para a inserção dos seguintes caracteres em uma Tabela Hash: ALGORITMOSEESTRUTURASDEDADOS. Use os métodos de de tratamento de colisão a seguir para tratar possíveis colisões.
    - Lista encadeada;
    - Endereçamento aberto;

Algoritmos e Estruturas de Dados III

- Bibliografia:
  - Básica:
    - ASCENCIO, Ana C. G. Estrutura de dados. Rio de Janeiro: Pearson. 2011.
    - CORMEN, Thomas; RIVEST, Ronald; STEIN, Clifford; LEISERSON, Charles. Algoritmos. Rio de Janeiro: Elsevier, 2002.
    - ZIVIANI, Nívio. Projeto de algoritmos com implementação em Pascal e C. São Paulo: Cengage Learning, 2010.
  - Complementar:
    - EDELWEISS, Nina, GALANTE, Renata. Estruturas de dados. Porto Alegre: Bookman. 2009. (Coleção Livros didáticos de informática UFRGS, 18).
    - PINTO, W.S. Introdução ao desenvolvimento de algoritmos e estrutura de dados. São Paulo: Érica, 1990.
    - PREISS, Bruno. Estruturas de dados e algoritmos. Rio de Janeiro: Campus, 2000.
    - TENEMBAUM. Aaron M. Estruturas de Dados usando C. São Paulo: Makron Books. 1995.
    - VELOSO, Paulo A. S. Complexidade de algoritmos: análise, projeto e métodos. Porto Alegre: Sagra Luzzatto, 2001.

Algoritmos e Estruturas de Dados III