UNIVERSIDADE ESTADUAL DE MARINGÁ
DEPARTAMENTO DE INFORMÁTICA

Transformação de Chave (*Hashing*)

Prof. Yandre Maldonado e Gomes da Costa
yandre@din.uem.br

Transformação de Chave - *Hashing*

- Transformações de Chave, ou Tabela *Hash*, ou Tabelas de Dispersão;
- "*Hashing*": fazer picadinho, ou fazer uma bagunça;
- Técnica que objetiva organizar um conjunto de dados, caracterizados por uma chave, de forma que o acesso tenha o menor custo possível;
- Em estruturas tradicionais (já estudadas), se os registros forem mantidos ordenados, algoritmos eficientes podem garantir busca à um custo **O(log n)**;

Transformação de Chave - *Hashing*

- Tabelas *hash*, se bem projetadas, podem garantir buscas à um custo constante **O(1)**;
- Os registros armazenados em uma tabela são diretamente endereçados a partir de uma transformação aritmética sobre a chave de pesquisa;
- O preço pago por esta eficiência será o uso maior de memória;

Transformação de Chave - *Hashing*

- Exemplo:
  - Suponhamos um sistema acadêmico que utiliza como chave o número de matrícula do aluno (RA) composto por **7 dígitos** numéricos;
  - Para permitir a busca à um registro de um aluno com um custo constante poderia ser utilizada a sua própria matrícula como índice de um vetor ***vet***;
  - Se isto fosse feito, poderíamos acessar os dados do aluno cuja matrícula é ***mat*** através de ***vet[mat]***;

Prof. Yandre Maldonado

4

Transformação de Chave - *Hashing*

- Assim, o acesso ocorre em ordem constante, entretanto o custo de memória para manter esse acesso é muito alto;
- Vamos considerar que o registro correspondente a cada aluno tenha a seguinte estrutura:

```
struct Aluno
 {
   int mat;
   char nome [80];
   char email [30];
   char turma;
 };
```

Prof. Yandre Maldonado

5

Transformação de Chave - *Hashing*

- Considerando que o número de matrícula é composto por **7 dígitos**, ele poderia ser caracterizado por qualquer número entre **0 e 9.999.999**;
- Assim, existe **10 milhões** de possíveis números de matrícula;
- Para isto, seria necessário um vetor com **10 milhões** de elementos, que poderia ser estabelecido por:

```
#define MAX 10000000
Aluno vet[MAX];
```

Prof. Yandre Maldonado

6

Transformação de Chave - *Hashing*

- Dessa forma, o nome do aluno de matrícula mat é acessado simplesmente através de **vet[mat].nome**;
- Embora o acesso seja rápido o custo de memória é proibitivo;
- Para a *struct* definida anteriormente teríamos um gasto de **115 bytes** de memória para cada aluno;
- Assim, o vetor descrito anteriormente consumiria **1.150.000.000 bytes**, ou seja, acima de **1 Gbyte** de memória;

Prof. Yandre Maldonado

7

Transformação de Chave - *Hashing*

- Em uma situação prática, seria comum encontrar um cadastro com algo em torno de **1.000 alunos**, o que necessitaria de apenas **115 Kbytes** de memória para armazenamento;

- Uma forma de resolver o problema do gasto excessivo de memória, garantindo um acesso rápido, é através do uso de tabela *hash*;

Prof. Yandre Maldonado

8

Transformação de Chave - *Hashing*

- Um método de pesquisa através de *hashing*, é constituído de duas etapas principais:
  - Computar o valor da função de transformação (função *hashing*), a qual transforma a chave de pesquisa em um endereço de tabela;
  - Considerando que duas ou mais chaves podem ser transformadas em um mesmo endereço de tabela, é necessário existir um método para lidar com **colisões**;

Prof. Yandre Maldonado

9

Transformação de Chave - *Hashing*

- Mesmo que o número de registros a serem armazenados seja muito menor do que o tamanho da tabela, qualquer que seja a função de transformação, fatalmente ocorrerão algumas **colisões**;
- Paradoxo do aniversário (Feller, 1968):

"Em um grupo de 23 ou mais pessoas juntas ao acaso, a probabilidade de que 2 pessoas comemorem aniversário no mesmo dia é maior do que 50%."

Prof. Yandre Maldonado

10

Transformação de Chave - *Hashing*

- Assim, em uma tabela com 365 endereços a probabilidade de colisão entre 23 chaves escolhidas aleatoriamente é maior do que 50%;
- Alta probabilidade de colisões, mesmo com distribuição uniforme;
- Exemplos:
  - Armazenar registros cuja chave é o RG de pessoas, com 7 dígitos numéricos, em uma tabela com 10000 endereços;
  - Armazenar registros cuja chave consiste de nomes compostos por até 16 letras (26^{16} chaves possíveis) em uma tabela com 1000 endereços;

Prof. Yandre Maldonado

11

Transformação de Chave - *Hashing*

- 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 ideal é aquela que:
    - seja simples de ser computada;
    - para cada chave de entrada, qualquer uma das saídas possíveis é igualmente provável de ocorrer.
  - Considerando que as transformações sobre as chaves são aritméticas, se houverem chaves não numéricas, elas devem ser transformadas em números;

Prof. Yandre Maldonado

12

Transformação de Chave - *Hashing*

- Um dos métodos que funciona muito bem para a transformação de chave, utiliza o resto da divisão por M:

$$h(K) = K \bmod M$$

  - Onde K é um inteiro correspondente à chave;
- Este é um método muito simples;
- Cuidado na escolha do valor de M:
  - Se M é par, então h(K) é par quando K é par, e h(K) é ímpar quando K é ímpar;
  - Assim, uma boa estratégia é escolher um valor primo para M;

Prof. Yandre Maldonado

13

Transformação de Chave - *Hashing*

- Um exemplo de função em linguagem C:

```
const n=10, M=7;
typedef char TipoChave[n];

int h(TipoChave Chave)
 {
  int i, Soma=0;
  for (i=0; i<n; i++)
   Soma += Chave[i];
  return (Soma % M);
 }
```

Prof. Yandre Maldonado

14

Transformação de Chave - *Hashing*

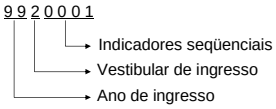
- Exercício:
  - Considere uma situação em que os itens de dados são identificados por uma chave composta por números inteiros de até 5 algarismos. Considere ainda que a função de transformação utilizada para endereçamento em uma tabela seja h(K)=K mod N, onde N é o tamanho da tabela. Considerando uma tabela de tamanho N=7, calcule os endereços para as seguintes chaves : 70013,10000, 7478, 197, 35019, 792, 10394, 5515, 8748, 15543.
  - Trace um gráfico mostrando a distribuição destas chaves na tabela.
  - Descreva a sua opinião a respeito da ocorrência (ou não) de colisões proporcionada pelas chaves descritas anteriormente e lance algumas hipóteses que possam viabilizar o tratamento de possíveis colisões em termos de estruturas de dados.

Prof. Yandre Maldonado

15

Transformação de Chave - *Hashing*

- Um estudo de caso:
  - Armazenar dados de alunos utilizando como chave o RA;
  - Alta probabilidade de **colisões**, mesmo com distribuição uniforme;
- Identificando **partes significativas da chave**:
  - Número de matrícula:



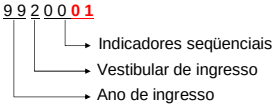
    - Pode-se adotar uma parte da chave de acordo com o tamanho da tabela que se pretende utilizar;

Prof. Yandre Maldonado

16

Transformação de Chave - *Hashing*

- Para armazenar apenas dados de alunos pertencentes à uma determinada turma:
  - Considerando que:
    - muitos alunos provavelmente ingressaram no mesmo ano;
    - muitos alunos provavelmente ingressaram no mesmo vestibular;
    - uma tabela de tamanho 100 seria suficiente para armazenar os dados dos alunos de uma turma;
  - Poderiam ser tomados os dois últimos dígitos dos indicadores seqüenciais para compor a chave;



Prof. Yandre Maldonado

17

Transformação de Chave - *Hashing*

- Assim, a tabela poderia ser dada por:

```
Aluno* tab[100];
```

- Neste caso, o acesso ao nome do aluno cujo número de matrícula é **mat** seria dado por:

```
vet[mat%100]->nome;
```

- A função de busca que mapeia uma chave para um índice da tabela poderia ser:

```
int hash (int mat)
 {
  return (mat%100);
 }
```

Prof. Yandre Maldonado

18

Transformação de Chave - *Hashing*

- De forma geral, para uma tabela de tamanho N, a função poderia ser dada por:

```
int hash (int mat)
 {
  return (mat%N);
 }
```

- Na prática, costuma-se utilizar um número primo para estabelecer o tamanho da tabela;
- Para minimizar o número de colisões, utiliza-se como regra empírica [Celes et al., 2004]:
  - Evitar taxa de ocupação superior a 75%;
  - Taxas em torno de 50%, em geral, trazem bons resultados;
  - Taxas menores do que 25% podem representar gasto excessivo de memória;

Prof. Yandre Maldonado

19

Transformação de Chave - *Hashing*

- Tratamento de colisões:
  - Vamos considerar um vetor de ponteiros, dado por:

```
#define N 127
typedef Aluno* Hash[N];
```

  - **Uso da posição consecutiva livre**:
    - Os elementos que colidem são armazenados em posições da tabela ainda não ocupadas;
    - Uma das estratégias consiste em buscar a **próxima posição livre** (utilizando incremento circular):

Prof. Yandre Maldonado

20

Transformação de Chave - *Hashing*

- Em uma busca, depois de mapeada a chave, procura-se a mesma no vetor até que ela ou uma posição vazia seja encontrada;

```
Aluno* hsh_busca (Hash tab, int mat)
 {
  int h=hash(mat);
  while (tab[h] != NULL) {
   if (tab[h]->mat == mat)
    return tab[h];
   h=(h+1)%N;
  }
  return NULL;
 }
```

Prof. Yandre Maldonado

21

Transformação de Chave - *Hashing*

- Uma função que insere ou modifica um elemento poderia ser dada por:

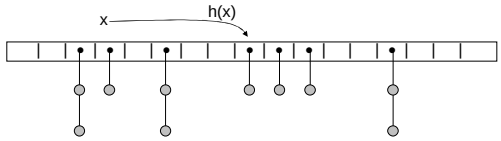
```
Aluno* hsh_insere (Hash tab, int mat, char n[81], char e[41], char t)
 {
  int h=hash(mat);
  while (tab[h] != NULL) {
   if (tab[h]->mat == mat)
    break;
   h=(h+1)%N;
  }
  if (tab[h]==NULL) {       //não encontrou o elemento
   tab[h] = (Aluno*) malloc (sizeof (Aluno));
   tab[h]->mat = mat;
  }
  strcpy (tab[h]->nome, n);   // atribui ou modifica informação
  strcpy (tab[h]->email, e);
  tab[h]->turma = t;
  return tab[h];
 }
```

Prof. Yandre Maldonado

Transformação de Chave - *Hashing*

– **Uso de listas encadeadas**:

- Estratégia diferente, consiste em fazer com que cada elemento da tabela *hash* represente um ponteiro para uma lista encadeada;



Prof. Yandre Maldonado

Transformação de Chave - *Hashing*

- Neste caso, a estrutura poderia ser dada por:

```
struct Aluno {
  int mat;
  char nome [80];
  char email [30];
  char turma;
  sturct Aluno* prox;
 };
```

Prof. Yandre Maldonado

Transformação de Chave - *Hashing*

- A operação de busca poderia ser descrita da seguinte forma:

```
Aluno* hsh_busca (Hash tab, int mat)
 {
  int h = hash(mat);
  Aluno* a = tab[h];
  while (a != NULL) {
   if (a->mat == mat)
    return a;
   a = a->prox;
  }
  return NULL;
 }
```

Prof. Yandre Maldonado

Transformação de Chave - *Hashing*

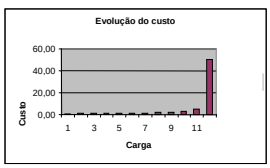
- Custo médio de uma consulta [Ziviani, 2002]:
  - Uso de posição consecutiva livre:
    - Considerando um fator de carga α dado por:
      - α=Q/N
        - » Onde Q é a quantidade de elementos inseridos na estrutura e N é o tamanho da tabela;
    - O custo médio é dado por:
      - Custo médio = ½ (1+1/(1- α))
  - Lista encadeada:
    - Custo médio = (1+ α)
      - Onde α=Q/N, e Q é a quantidade de elementos inseridos na estrutura e N é o tamanho da tabela;
      - A constante 1 corresponde ao tempo gasto para cálculo da função *hash*;

Prof. Yandre Maldonado

Transformação de Chave - *Hashing*

- Evolução do custo médio para posição consecutiva livre:

	Q	N	Custo
1	10	100	1,06
2	20	100	1,13
3	**25**	**100**	**1,17**
4	30	100	1,21
5	40	100	1,33
6	**50**	**100**	**1,50**
7	60	100	1,75
8	70	100	2,17
9	**75**	**100**	**2,50**
10	80	100	3,00
11	90	100	5,50
12	99	100	50,50



Prof. Yandre Maldonado

Transformação de Chave - *Hashing*

- Referências Bibliográficas:
  - Celes, Waldemar et al. **Introdução a Estruturas de Dados**. Rio de Janeiro: Elsevier, 2004;
  - Pereira, Silvio do Lago. **Estruturas de Dados Fundamentais**. São Paulo: Érica, 1996;
  - Wirth, Niklaus. **Algoritmos e Estruturas de Dados**. Rio de Janeiro: LTC, 1999;
  - Ziviani, Nivio. **Projeto de Algoritmos – com implementações em Pascal e C**. São Paulo: Pioneira Thomson Learning, 2002;

Prof. Yandre Maldonado