Pesquisa em Memória Primária

Hashing

Livro “Projeto de Algoritmos” – Nívio Ziviani

Capítulo 5 – Seções 5.5

http://www2.dcc.ufmg.br/livros/algoritmos/

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 (Webster's New World Dictionary)
  - Fazer picadinho de carne e vegetais para cozinhar.
  - Fazer uma bagunça.

Transformação de Chave (Hashing)

- "Hash", em computação, tem outros significados e outros usos
  - Em criptografia: hash é o resultado de algoritmos de dispersão, que produzem um código binário a partir de um arquivo
    - MD5 (Rivest (RSA), 1992): 128 bits
    - Usado para verificar a integridade do arquivo após transmissão, combinado com alguma forma de checksum
  - Twitter: "hashtag" é uma marca que permite ao Twitter acompanhar as tendências nas discussões e assuntos relevantes
    - Ex.: (2009) #forasarney, #classicmoviequotes
- **Significado geral: transformar algo grande em algo pequeno**

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.

Transformação de Chave (Hashing)

- 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)

- A probabilidade p de se inserir N itens consecutivos sem colisão em uma tabela de tamanho M é:

$$p = \frac{M-1}{M} \times \frac{M-2}{M} \times \ldots \times \frac{M-N+1}{M} = \prod_{i=1}^{N} \frac{M-i+1}{M} = \frac{M!}{(M-N)!M^N}$$

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 ≈ 87,7%.

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 (em linguagem C; em Pascal h(K) = K mod M)

  onde K é um inteiro correspondente à chave.

Transformação 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 ≤ i ≤ n.

Transformação 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 Chaves Não Numéricas

Implementação da função de transformação:

```
Indice h(TipoChave Chave, TipoPesos p)
{
  int i;
   unsigned int Soma = 0;
   int comp = strlen(Chave);

   for (i = 0; i < comp; i++)
     Soma += (unsigned int)Chave[i] * p[i];

    return (Soma % M);
}
```

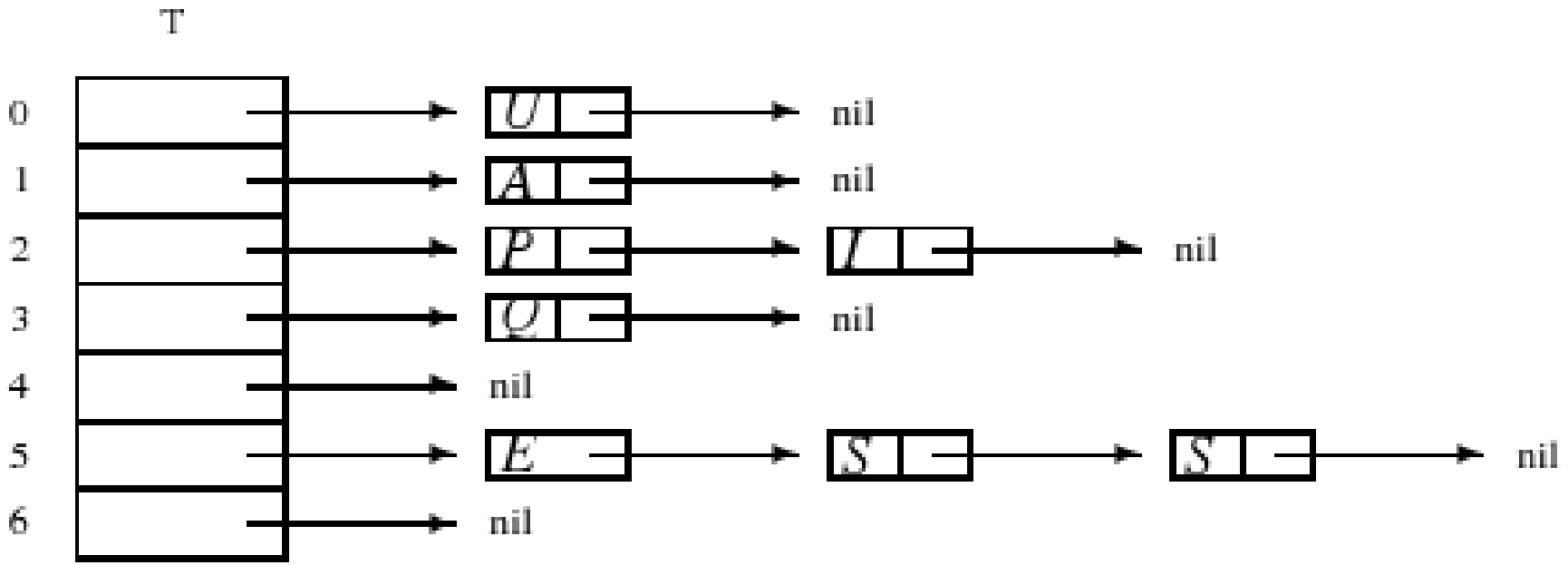
Listas Encadeadas

Uma das formas de resolver as **colisões** é simplesmente construir uma lista linear encadeada para cada endereço da tabela. Assim, todas as chaves com mesmo endereço são encadeadas em uma lista linear.

Listas Encadeadas

- **Exemplo:** 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 P E S Q U I S A na tabela é o seguinte:

- Por exemplo, h(A) = h(1) = 1, h(E) = h(5) = 5, h(S) = h(19) = 5, etc



Estrutura do Dicionário para Listas Encadeadas

```
#define M                  7
#define n                  10
typedef char TipoChave[n];

typedef unsigned int TipoPesos[n];

typedef struct {
  /* outros componentes */
  TipoChave Chave;
} TipoItem;

typedef unsigned int Indice;
```

Estrutura do Dicionário para Listas Encadeadas

```
typedef struct _celula* Apontador;

typedef struct _celula {
  TipoItem Item;
  Apontador Prox;
} Celula;

typedef struct
{
   Celula *Primeiro, *Ultimo;
} TipoLista;

typedef TipoLista TipoDicionario[M];
```

Operações do Dicionário Usando Listas Encadeadas

```
void Inicializa(TipoDicionario T)
{
  int i;
  for (i = 0; i < M; i++)
     FLVazia(&T[i]);
}
```

Operações do Dicionário Usando Listas Encadeadas

```
Apontador Pesquisa(TipoChave Ch, TipoPesos p, TipoDicionario T)
{
/*Obs: Apontador de retorno aponta para o item anterior da lista */
  Indice i;   Apontador Ap;
  i = h(Ch, p);
  if (Vazia(T[i])) return NULL;   /* Pesquisa sem sucesso */
  else  {
     Ap = T[i].Primeiro;
     while ((Ap->Prox->Prox != NULL) &&
           (strncmp(Ch, Ap->Prox->Item.Chave, sizeof(TipoChave)) ))
       {Ap = Ap->Prox;}
       if (!strncmp(Ch, Ap->Prox->Item.Chave, sizeof(TipoChave)))
          return Ap;
       else return NULL;   /* Pesquisa sem sucesso */
    }
}
```

Operações do Dicionário Usando Listas Encadeadas

```
void Insere(TipoItem x, TipoPesos p, TipoDicionario T)
{
   if (Pesquisa(x.Chave, p, T) == NULL)
      Ins(x, &T[h(x.Chave, p)]); /* Insere do TAD Lista */
   else
      printf(" Registro ja  esta  presente\n");
}

void Retira(TipoItem x, TipoPesos p, TipoDicionario T)
{
  Apontador Ap;

  Ap = Pesquisa(x.Chave, p, T);

  if (Ap == NULL)
    printf(" Registro nao esta  presente\n");
  else
    Ret(Ap, &T[h(x.Chave, p)], &x); /* Retira do TAD Lista */

}
```

Análise

- Assumindo que qualquer item do conjunto tem igual probabilidade de ser endereçado para qualquer entrada de T, então o comprimento esperado de cada lista encadeada é N/M, onde N representa o número de registros na tabela e M o tamanho da tabela.
- **Logo:** as operações Pesquisa, Insere e Retira custam O(1 + N/M) operações em média, onde a constante 1 representa o tempo para encontrar a entrada na tabela e N/M o tempo para percorrer a lista.

  Para valores de M próximos de N , o tempo se torna constante, isto é, independente de N .

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)

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:

$$h_j = (h(x) + j) \bmod M, \text{ para } 1 \leq j \leq M - 1.$$

Exemplo

Se a i-ésima letra do alfabeto é representada pelo número i e a função de transformação

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

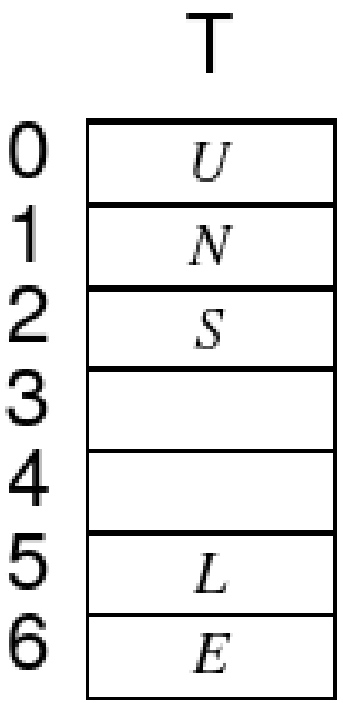
é utilizada para M = 7,

então o resultado da inserção das chaves L U N E S na tabela, usando hashing linear para resolver colisões é mostrado abaixo.

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.



Estrutura do Dicionário Usando Endereçamento Aberto

```
#define Vazio          "!!!!!!!!!!"
#define Retirado       "**********"
#define M              7
#define n              11  /* Tamanho da chave */
```

Estrutura do Dicionário Usando Endereçamento Aberto

```
typedef unsigned int Apontador;

typedef char TipoChave[n];
typedef unsigned TipoPesos[n];

typedef struct {
  /* outros componentes */
  TipoChave Chave;
} TipoItem;

typedef unsigned int Indice;
typedef TipoItem TipoDicionario[M];
```

Operações do Dicionário Usando Endereçamento Aberto

```
void Inicializa(TipoDicionario T)
{
   int i;
   for (i = 0; i < M; i++)
     memcpy(T[i].Chave, Vazio, n);
}
```

Operações do Dicionário Usando Endereçamento Aberto

```
Apontador Pesquisa(TipoChave Ch, TipoPesos p, TipoDicionario T)
{ unsigned int  i = 0;
  unsigned int  Inicial;

  Inicial = h(Ch, p);

   while ( (strcmp (T[(Inicial + i) % M].Chave,Vazio) != 0) &&
           (strcmp ( T[(Inicial + i) % M].Chave, Ch) != 0)  &&
           (i < M))
               i++;
  if (strcmp (T[(Inicial + i) % M].Chave, Ch) == 0)
     return ((Inicial + i) % M);
  else return M;   /* Pesquisa sem sucesso */
}
```

Operações do Dicionário Usando Endereçamento Aberto

```
void Retira(TipoChave Ch, TipoPesos p, TipoDicionario T)
{
   Indice i;

   i = Pesquisa(Ch, p, T);
   if (i < M)
     memcpy(T[i].Chave, Retirado, n);
   else
     printf("Registro nao esta presente\n");
}
```

Operações do Dicionário Usando Endereçamento Aberto

```
void Insere(TipoItem x, TipoPesos p, TipoDicionario T)
{
  unsigned int i = 0;
  unsigned int Inicial;

  if (Pesquisa(x.Chave, p, T) < M)  {
    printf("Elemento ja esta presente\n");
    return;
  }
  Inicial = h(x.Chave, p);
  while ((strcmp(T[(Inicial + i) % M].Chave,Vazio) != 0) &&
         (strcmp(T[(Inicial + i) % M].Chave,Retirado) != 0) &&
         (i < M))
           i++;
  if (i < M)   {
     /* Copiar os demais campos de x, se existirem */
     strcpy (T[(Inicial + i) % M].Chave, x.Chave);
  }
  else printf(" Tabela cheia\n");
}
```

Análise

- Seja α = N / M o fator de carga da tabela. Conforme demonstrado por Knuth (1973), o custo de uma pesquisa com sucesso é

$$C(n) = \frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$$

- O hashing linear sofre de um mal chamado agrupamento(clustering) (Knuth, 1973, pp.520–521).

- Este fenômeno ocorre na medida em que a tabela começa a ficar cheia, pois a inserção de uma nova chave tende a ocupar uma posição na tabela que esteja contígua a outras posições já ocupadas, o que deteriora o tempo necessário para novas pesquisas.

Análise

- Entretanto, apesar do hashing linear ser um método relativamente pobre para resolver colisões os resultados apresentados são bons.

- O melhor caso, assim como o caso médio, é O(1).

Vantagens e Desvantagens de Transformação da Chave

- Vantagens:
  - • Alta eficiência no custo de pesquisa, que é O(1) para o caso médio.
  - •Simplicidade de implementação

- Desvantagens:
  - • Custo para recuperar os registros na ordem lexicográfica das chaves é alto, sendo necessário ordenar o arquivo.
  - • Pior caso é O(N)