Estruturas de Dados

Tabelas Hash

Baseado no material dos Profs:
Carlos F. S. Costa
Rodolfo Pereira

Prof. Nilton Luiz Queiroz Jr.

Pesquisa

- Alguns métodos de pesquisa em vetor são baseados na comparação da chave pesquisada com as chaves armazenadas;
  - Busca sequencial;
    - Vetor não ordenado;
    - Pior caso custo O(n);
  - Busca binária;
    - Vetor ordenado;
    - Pior caso custo $O(\log_2 n)$;
- Porém, o acesso a um elemento do vetor é feito em tempo constante;
- Existe uma maneira organizar os dados para que possa se recuperar a informação em tempo constante?
  - O(1);

Tabelas Hash

- Dado uma chave, podemos aplicar uma função que transforme a chave em um índice válido para o vetor;
  - Essa função é chamada de função Hash (ou função de dispersão);
- As tabelas Hash são um tipo de estrutura criado para o armazenamento de informações e são:
  - Simples de implementar;
  - Intuitivas para se organizar grandes quantidades de dados;

Tabelas Hash

- A maneira que as tabelas são estruturadas visa, principalmente, permitir armazenar e procurar em grandes quantidades de dados;
- A ideia central é dividir o universo de informações em subconjuntos mais facilmente gerenciáveis;
  - Em outras palavras, dividimos um conjunto de dados C em um número finito de partes C_1, C_2, C_3, ..., C_n tais que:
    - $C_1 \cup C_2 \cup C_3 \cup \ldots \cup C_n = C$
    - $C_i \cap C_j = \emptyset \ \forall i,j \in [1,n]$

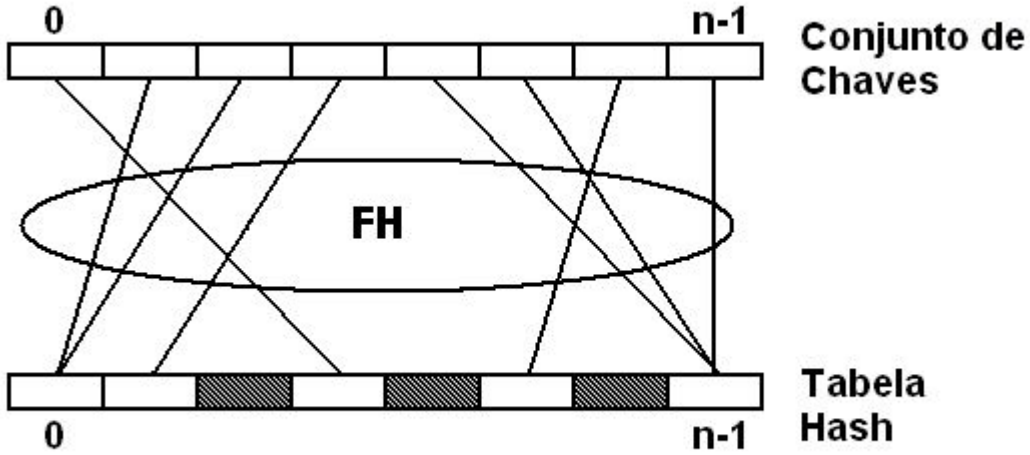
Tabelas Hash

- Alguns conceitos fundamentais para o estudo de Hashs:
  - Tabela Hash: Estrutura que permite o acesso aos subconjuntos;
  - Função Hash: Função que realiza um mapeamento entre os valores de chaves e as entradas na tabela;
- Os registros armazenados em uma tabela hash são diretamente endereçados a partir de uma função hash;

Funções Hash

- As funções Hash tem como objetivo mapear elementos em uma tabela Hash, porém, dois elementos podem ser levados ao mesmo lugar da tabela!
- Quando dois elementos tentam ocupar o mesmo lugar na tabela temos uma **colisão** entre eles;
- Dessa maneira, ao implementar uma tabela hash deve-se tratar as colisões;

Funções Hash



Tabelas Hash

- A pesquisa por meio de funções hash é feita em duas etapas:
  - Computar o valor da função de transformação (também conhecida por função hashing), a qual transforma a chave de pesquisa em um endereço da 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.

Funções Hash

- Existem diversas funções hash implementadas a literatura;
  - Uma função simples é o método da divisão:
    - Dada uma chave, obtém-se o resto da divisão da chave pelo tamanho do hash;
    - $f(k) = k \bmod m$;
      - Sendo k a chave do elemento que será inserido numa tabela contendo elementos das posições 0 à m-1;
    - Alguns autores trazem como $f(k) = (k \bmod m)+1$;
  - Para implementar o método da divisão é necessário um cuidado:
    - O valor de m;
      - Em geral escolhe-se m como sendo um número primo, **evitando** números primos obtidos por meio de $\mathbf{b^i \pm j}$
        - Onde b é a base do conjunto de caracteres (128 para ASCII, por exemplo) i e j são pequenos inteiros;

Tabelas Hash

- Exemplo do método da divisão:
  - Usando o método da divisão para m = 4, esquematize uma tabela Hash T, inicialmente vazia, após a inserção dos elementos: 13, 27, 38, 40, 52, 64, 71, 88 e 90.
  - T={13, 27, 38, 40, 52, 64, 71, 88, 90}
  - Como o tamanho da tabela é 4 teremos 4 subconjuntos;
  - Resultado:
    - $f(13) = 13 \bmod 4 = 1$;
    - $f(27) = 27 \bmod 4 = 3$;
    - $f(38) = 38 \bmod 4 = 2$;
    - $f(40) = 40 \bmod 4 = 0$
    - $f(52) = 52 \bmod 4 = 0$;
    - $f(64) = 64 \bmod 4 = 0$
    - $f(71) = 71 \bmod 4 = 3$;
    - $f(88) = 88 \bmod 4 = 0$;
    - $f(90) = 90 \bmod 4 = 2$;

$T_0 = \{40, 52, 64, 88\}$

$T_1 = \{13\}$

$T_2 = \{38, 90\}$

$T_3 = \{27, 71\}$

Colisões

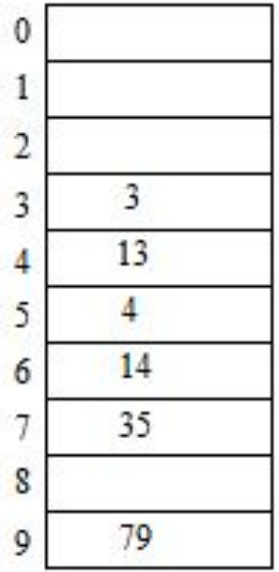
- Como visto no exemplo anterior, podem ocorrer situações em que mais de um elemento precise ocupar o mesmo local no hash, ocorrendo então uma colisão;
- Assim, para tratar colisões existem duas filosofias de Hash:
  - Reespalhamento;
  - Listas ligadas;

Reespalhamento

- Tem um espaço fixo para inserir elementos;
- Usa-se uma função secundária sobre a chave;
  - Exemplo, se existir somente uma posição para cada subconjunto:
    - Se ocorrer uma colisão, tente fazer a inserção na posição *f*(k) + 1;
    - Caso ocorra uma colisão na posição *f*(k) + 1 tente inserção em *f*(k) + 2;
    - Prossiga até encontrar uma posição;

Exemplo

- Dada a função hash *f*(x) = x mod 10, mostre a tabela após a inserção dos valores 3, 13, 4, 14, 35,79.



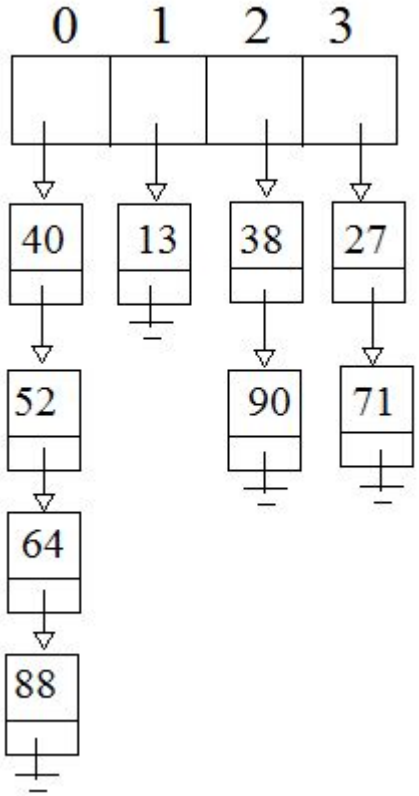
Lista ligada

- Usa-se uma lista ligada para tratar colisões;
- Quando ocorrem colisões os elementos deverão ser então armazenados na lista ligadaque é “armazenada” pelo elemento do hash que seria o responsável por armazenar os dois elementos;
- Em hashs com lista ligada deve-se agir da seguinte maneira na pesquisa:
  - Calcular qual elemento de entrada do array é o ponteiro que “armazena” a estrutura;
  - Utiliza-se a técnica de busca adequada para a lista;

Exemplo

- Usando o método da divisão para m = 4, mostre a estrutura de uma tabela Hash T, inicialmente vazia, após a inserção dos elementos: 13, 27, 38, 40, 52, 64, 71, 88 e 90 nessa ordem.

Exemplo



Vantagens das Tabelas Hash

- São simples e de fácil implementação;
- São eficientes para grande número de elementos;
    - Supondo um problema com cem mil elementos, poderíamos imaginar uma tabela hash com mil entradas, onde temos uma divisão de espaço de procura da ordem de 100000/1000 de imediato;

Desvantagens da Tabela Hash

- Dependência da função Hash;
  - Para que o tempo de acesso seja mantido, a função hash deve distribuir os elementos de entrada de maneira igual;
    - A função deve distribuir os elementos maneira que cada subconjunto tenha sempre uma quantidade próxima dos demais;
  - Funções que não distribuem bem os elementos podem tornar os hash com eficiência próxima a estrutura de dados auxiliar usada;

Implementação

- A implementação de um hash pode ser feita sobre uma lista;
  - Seja ela simplesmente ou duplamente ligada, circular ou linear, com ou sem sentinela;
- Dessa maneira, a tabela será uma tabela de listas, e a função hash irá decidir em que lista deve-se operar;

Implementação

- A implementação a seguir leva em conta que já se tem implementada uma lista ligada que armazena os dados do tipo que será também armazenado na tabela hash;
- Serão implementadas as seguintes operações:
  - Inserção na tabela hash;
  - Remoção da tabela hash;
  - Busca na tabela hash;
- Além dessas 3 operações será necessária a implementação de uma função hash e também uma operação para inicializar a tabela;

Implementação

- Para as operações sobre listas assuma o seguinte:
  - inicializa_lista(l);
    - Inicializa a lista l;
  - buscar_valor_lista(l,ch,i);
    - Procura pelo elemento de chave ch na lista l:
      - Caso exista, **i** é alterado para o item de chave ch e a função retorna 1;
      - Caso contrário retorna 0;
  - insere_ultimo(l,i);
    - Procura pela chave do elemento i na lista l;
      - Caso não exista, insere **i** no final da lista l e a função retorna 1;
      - Caso contrário retorna 0;
  - remove_lista_chave(l,ch,i);
    - Procura pela chave do elemento ch na lista l;
      - Caso exista, **i** é alterado para o item de chave ch e a função retorna 1;
      - Caso contrário retorna 0;

Implementação

- Para implementarmos um hash, além dos tipos usados na lista, usa-se também as seguintes declarações de constantes e tipo:

```
#include "lista.h" // cabeçalho contendo todas as implementações de funções de listas
#include <stdio.h>

#define TAM 13

struct tipo_hash{
    struct tipo_lista tabela[TAM];
};
```

Implementação

- Antes de começar a inserir e remover elementos na tabela é necessário inicializá-la;

```
void inicializa_hash(struct tipo_hash *h){
    /*algoritmo*/
}
```

Implementação

- Antes de começar a inserir e remover elementos na tabela é necessário inicializá-la;

```
void inicializa_hash(struct tipo_hash *h){
    int i;
    for(i=0;i<TAM;i++){
        inicializa_lista(&h->tabela[i]);
    }
}
```

Implementação

- A função hash define em qual lista será armazenado cada elemento

```
int funcao_hash(int chave){
    /*algoritmo*/
}
```

Implementação

- A função hash define em qual lista será armazenado cada elemento

```
int funcao_hash(int chave){
    return chave%TAM;
}
```

Implementação

- A inserção em um hash requer a inserção em lista;
  - Cada elemento será armazenado na lista que é indexada pelo valor da função hash para sua chave;

```
int insere_hash(struct tipo_hash *h, struct tipo_item x){
    /*algoritmo*/
}
```

Implementação

- A inserção em um hash requer a inserção em lista;
  - Cada elemento será armazenado na lista que é indexada pelo valor da função hash para sua chave;

```
int insere_hash(struct tipo_hash *h, struct tipo_item x){
    int idx = funcao_hash(x.chave);
    return insere_ultimo(&h->tabela[idx],x);
}
```

Implementação

- Busca

```
int buscar_hash(struct tipo_hash *h, int ch, struct tipo_item *x){
    /*algoritmo*/
}
```

Implementação

- Busca

```
int buscar_hash(struct tipo_hash *h, int ch, struct tipo_item *x){
    int idx = funcao_hash(ch);
    return buscar_valor_lista(&h->tabela[idx],ch,x);
}
```

Implementação

- Remoção;

```
int remove_hash(struct tipo_hash *h, int ch, struct tipo_item *x){
    /**/
}
```

Implementação

- Remoção;

```
int remove_hash(struct tipo_hash *h, int ch, struct tipo_item *x){
    int idx = funcao_hash(ch);
    return remove_lista_chave(&h->tabela[idx],ch,x);
}
```

Exercícios

1. Mostre o hash resultante após a inserção de cada elemento tratando colisões com listas ligadas, e tendo como função de hash a função:

   f(k) = k mod 9;

   1, 9, 12, 7, 3, 18

2. Faça o exercício 1 para um hash fechado que trata as colisões colocando o elemento na posição seguinte;
3. Implemente um subprograma que mostre o estado do hash, ou seja, qual o índice da lista e quais os valores armazenados nela;
   OBS: Mantenha a modularidade dos arquivos, ou seja, se for usar funções para a lista, implemente-as no arquivo da lista.

Referências

ZIVIANI, N.; Projeto de Algoritmos com Implementações em Pascal e C, 3ª Edição, Livraria Pioneira Editora, 1996.