Ciência da Computação
Campus Arapiraca

Aula 08: Tabelas de Hash

Prof. Dr. Rodolfo Carneiro
rodolfo.cavalcante@arapiraca.ufal.br

Introdução

- Até agora estudamos estruturas de dados lineares
  - Valores se sucedem em sequência
  - Existe uma ordenação topológica dos dados
- A ordenação pode tornar operações de inserção, remoção ou consulta custosas
  - Tipicamente O(n)
- Se temos muitos dados, o ideal seria tempo constante

Introdução

- Hoje vamos estudar uma estrutura de dados diferente
  - As tabelas hashing
- Generalização de um vetor com *m* posições
- Os elementos a serem armazenados possuem um valor-chave
  - Utilizado para calcular o endereço da tabela onde serão armazenados

Introdução

- Vejam o seguinte exemplo:
  - Existe um país pequeno com população de **bem menos** de 100 mil habitantes
  - O CPF nesse país tem 5 dígitos
  - Poderíamos construir a seguinte tabela:

chave	valor
01555	Ronaldo
01567	Pelé
...	...
80320	Dunga
95222	Romário

Introdução

- Vejam o seguinte exemplo:
  - Existe um país pequeno com população de **bem menos** de 100 mil habitantes
  - O CPF nesse país tem 5 dígitos
  - Decisão mais fácil de estrutura de armazenamento:
    - Vetor de tamanho 100 mil
    - A chave é o índice do vetor [0 a 99999]
    - Inserção, remoção e consulta em O(1)
    - Problema?

Introdução

- Vejam o seguinte exemplo:
  - Decisão mais fácil de estrutura de armazenamento:
    - Vetor de tamanho 100 mil
    - A chave é o índice do vetor
    - Inserção, remoção e consulta em O(1)
  - Problema?
    - Muitas posições vagas
    - Custo de espaço

Introdução

- Vejam o seguinte exemplo:
  - Outra possível solução:
    - Usar o nome como chave
    - Usar o cpf como valor
    - Organizar em ordem alfabética
  - Custos para inserção, remoção e consulta?

Introdução

- Vejam o seguinte exemplo:
  - Outra possível solução:
    - Usar o nome como chave e usar o cpf como valor
    - Organizar em ordem alfabética
  - Custos para inserção, remoção e consulta?
  - Para acelerar a busca, podemos divider a lista pela letra inicial (26 partes)
    - O vetor de 26 posições é a tabela de hash
    - Cada posição do vetor aponta para o começo da lista de nomes

Tabela de Hash

- Chamada de tabela de dispersão
- Função de hash
  - Transforma uma chave x em um índice da tabela h(x)
  - Espalha as chaves pela tabela de hash
  - No caso do cpf como chave, a função coloca o valor no índice igual ao cpf
  - No caso do nome como chave, a função mapeia a primeira letra para uma lista de valores

Tabela de Hash

- Definição
  - Sejam um conjunto *C* de chaves a serem armazenadas
  - Seja uma tabela de tamanho *m*
  - Seja x uma chave em C
  - Existe uma função h(x) que mapeia x para um endereço m = {0,…,m-1}

Tabela de Hash

- A maioria das funções de hash trabalha com elementos-chave que são números naturais
  - Em caso de strings como chave, pode-se utilizar o número natural que representa a soma dos códigos ASCII dos caracteres
- Proriedades
  - Unidirecional: não se recupera a chave *x* a partir do resultado da função *h(x)*
  - Colisão: duas chaves diferentes tem o mesmo valor de hash

Tabela de Hash

- Exemplo: Método da divisão
- Chave *x* mapeada para o endereço que é o resto da divisão de *x* por *m*
  - h(x) = x % m
- Ex: tabela com *m* = 8, chave *x* = 100
  - H(100) = 4
- Ex: tabela com *m* = 8, chave *x* = 52
  - H(52) = 4
- Colisão

Tabela de Hash

- Segundo Cormem (2002) função de hashing deve satisfazer as seguintes condições
  - Gerar número pequeno de colisões
  - Ser fácil de computar (rápido)
  - Ser uniforme: probabilidade de resultar em qualquer endereço k = [0,m-1] é a mesma para todas as chaves
- Nem sempre é possível minimizar as colisões
  - Métodos de tratamento de colições

Implementação

- Existem algumas formas de implementar
  - Endereçamento aberto
  - Implementação como lista

Endereçamento Aberto

- Tabela é vetor com *m* posições
- Chaves armazenadas na tabela
  - Sem necessidad de espaços extras ou ponteiros
- Posições vazias da tabela são utilizadas para o tratamento de colisões
  - chave *x* é endereçada na posição *h(x)*, mas está ocupada
  - Próxima posição vazia na tabela é procurada
  - Caso não haja posição vazia, tabela está cheia

Endereçamento Aberto

- Quando ocorrer uma colisão utiliza-se a seguinte função
  - h’(x) = (h(x)+j)mod m
  - Para 1 ≤ j ≤ m-1
- Ex: tabela com *m* = 8,
  - chave *x* = 100 inserido em h(100) = 4
  - chave x = 94 inserido em h(94) = 6
  - chave *x* = 52 inserido em h(52) = 4+1 = 5

Endereçamento Aberto

- A remoção será feita da mesma forma
  - Usa-se h(x) para identificar quem será removido
- Problema: como tratar a colisão na remoção?
  - Remover x = 52?

Endereçamento Aberto

- Problema: como tratar a colisão na remoção?
  - Remover x = 52?
- Solução
  - Marcar cada endereço como livre (L), ocupado (O) ou removido (R)
  - Livre: quando nunca recebeu valor
  - Ocupado: quando está com valor
  - Removido: já possuiu valor e agora não mais
    - Usado para resolver a colisão na remoção

Endereçamento Aberto

- Criar tabela

Endereçamento Aberto

- Criar tabela

Endereçamento Aberto

- Inserir x = 16

Endereçamento Aberto

- Inserir x = 16

	O	16
0	L	
1	L	
2	L	
3	L	
4	L	
5	L	
6	L	
7	L	

Endereçamento Aberto

- Inserir x = 23

	O	16
0		
1	L	
2	L	
3	L	
4	L	
5	L	
6	L	
7	L	

Endereçamento Aberto

- Inserir x = 23

0	O	16
1	L	
2	L	
3	L	
4	L	
5	L	
6	L	
7	O	23

Endereçamento Aberto

- Inserir x = 41

0	O	16
1	L	
2	L	
3	L	
4	L	
5	L	
6	L	
7	O	23

Endereçamento Aberto

- Inserir x = 41

0	O	16
1	O	41
2	L	
3	L	
4	L	
5	L	
6	L	
7	O	23

Endereçamento Aberto

- Inserir x = 25

0	O	16
1	O	41
2	L	
3	L	
4	L	
5	L	
6	L	
7	O	23

Endereçamento Aberto

- Inserir x = 25

0	O	16
1	O	41
2	O	25
3	L	
4	L	
5	L	
6	L	
7	O	23

Endereçamento Aberto

- Inserir x = 39

0	O	16
1	O	41
2	O	25
3	L	
4	L	
5	L	
6	L	
7	O	23

Endereçamento Aberto

- Inserir x = 39

0	O	16
1	O	41
2	O	25
3	O	39
4	L	
5	L	
6	L	
7	O	23

Endereçamento Aberto

- Remover x = 41

0	O	16
1	O	41
2	O	25
3	O	39
4	L	
5	L	
6	L	
7	O	23

Endereçamento Aberto

- Remover x = 41

0	O	16
1	R	
2	O	25
3	O	39
4	L	
5	L	
6	L	
7	O	23

Endereçamento Aberto

- Remover x = 23

0	O	16
1	R	
2	O	25
3	O	39
4	L	
5	L	
6	L	
7	O	23

Endereçamento Aberto

- Remover x = 23

0	O	16
1	R	
2	O	25
3	O	39
4	L	
5	L	
6	L	
7	R	

Endereçamento Aberto

- Remover x = 25

0	O	16
1	R	
2	O	25
3	O	39
4	L	
5	L	
6	L	
7	R	

Endereçamento Aberto

- Remover x = 25

0	O	16
1	R	
2	R	
3	O	39
4	L	
5	L	
6	L	
7	R	

Endereçamento Aberto

- Inserir x = 34

0	O	16
1	R	
2	R	
3	O	39
4	L	
5	L	
6	L	
7	R	

Endereçamento Aberto

- Inserir x = 34

0	O	16
1	R	
2	O	34
3	O	39
4	L	
5	L	
6	L	
7	R	

Endereçamento Aberto

- Implementação

Endereçamento Aberto

- Análise de complexidade
  - O custo de computar a tabela hash é uma função artimética
  - Custo da colisão?

Tabela Implementada como Lista

- Uma outra forma de resolver o problema de colisão é implementando a tabela com lista
- A tabela hashing é um vetor de *m* posições
- Cada posição possui um ponteiro para uma lista encadeada
  - Que guarda os valores que possuem mesmo endereço mapeado
- Quando houver colisão, basta inserir elemento na lista do endereço

Tabela Hashing com Lista

- Implementação