Tabela Hash (hashing)

Manassés Ribeiro
manasses.ribeiro@ifc.edu.br

Agenda

- Conceitos preliminares
- Tabela de espalhamento (hashing)
- Funções de espalhamento (hashing)
- Tratamento de colisões

Conceitos preliminares

Problema 1: imagine um pequeno país (bem menos que 100 mil habitantes) onde os números de CPF têm apenas 5 dígitos decimais. Considere a tabela que leva CPFs em nomes

chave	valor associado
01555	Ronaldo
01567	Pelé
...	...
80114	Maradona
80320	Dunga
95222	Romário

Conceitos preliminares

- Como armazenar a tabela?

Conceitos preliminares

- Como armazenar a tabela?
  - **Resposta óbvia: vetor de 100 mil posições**;
  - Use a própria chave como índice do vetor!

Conceitos preliminares

- Como armazenar a tabela?
  - **Resposta óbvia: vetor de 100 mil posições**;
  - Use a própria chave como índice do vetor!
- O vetor é conhecido com **tabela de hash** e terá **muitas posições vagas** (desperdício de espaço)
  - apesar da busca (get) e da inserção (put) serem bastante rápidas.

Conceitos preliminares

Problema 2: imagine uma lista ligada onde as chaves são nomes de pessoas. Suponha que a lista está em ordem alfabética.

chave	valor associado
Antonio Silva	8536152
Arthur Costa	7210629
Bruno Carvalho	8536339
...	...
Vitor Sales	8535922
Wellington Lima	5992240
Yan Ferreira	8536023

Conceitos preliminares

- Para acelerar as buscas, pode-se dividir a lista em 26 pedaços:
  - os nomes que começam com "A", os que começam com "B", etc.
- Nesse caso, o vetor de 26 posições é a tabela de hash e cada posição do vetor aponta para o começo de uma das listas.

Tabela Hash (hashing)

- É uma tabela de dispersão
  - também conhecida como tabela de hash (hash table)
  - é um vetor onde cada uma das posições armazena zero, uma, ou mais chaves (e valores associados);
  - o conceito é propositalmente vago.
- Hashing tem dois ingredientes fundamentais:
  - uma função de hashing; e
  - um mecanismo de resolução de colisões.

Tabela Hash (hashing)

- Parâmetros importantes:
  M: número de posições na tabela de hash
  N: número de chaves da tabela de símbolos
  α **= N/M:** fator de carga (*load factor*)

Função de espalhamento (função de hashing)

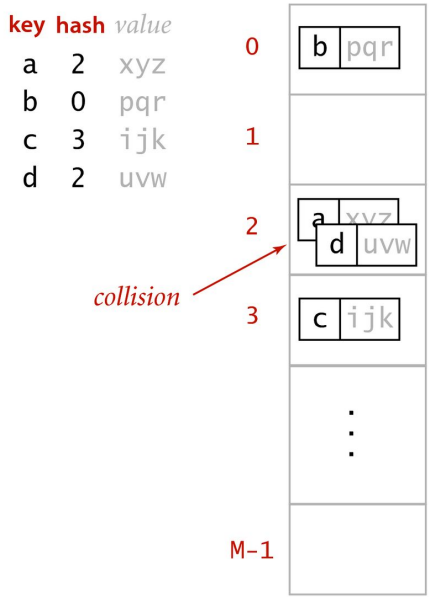
- **Transforma cada chave** em um índice da tabela de hash.
- A **função de hashing** responde a pergunta:
  - em qual **posição da tabela de hash** deve ser colocada determinada chave?
  - **espalha as chaves** pela tabela de hash.

Função de espalhamento (função de hashing)

- A função de hashing **associa um valor hash** (*hash value*), entre 0 e M−1, a cada chave:
  - no exemplo dos CPFs tem-se $\alpha < 1$ e a função de hashing é a identidade;
  - no exemplo dos nomes de pessoas tem-se $\alpha > 1$ e a função de hashing é o primeiro caractere do nome.

Função de espalhamento (função de hashing)

- A função de hashing produz uma **colisão** quando duas chaves diferentes têm o **mesmo valor hash** e portanto são levadas na mesma posição da tabela de hash



Hashing: the crux of the problem

Função de espalhamento (função de hashing)

Exemplo: chaves são números de identificação (7 dígitos) de estudantes do IFC e M é 100.

- Possíveis funções de hashing:
  - 2 primeiros dígitos da chave, os 2 dígitos do meio ou os 2 últimos dígitos.
- **Qual dessas opções espalha melhor as chaves pelo intervalo 0 . . 99?**

8536152
7210629
8536339
8536002
...
8067490
8536106
8536169
8531845

Função de espalhamento (função de hashing)

- O ideal é que a função de hashing **use todos os dígitos** da chave;
  - assim, chaves ligeiramente diferentes serão levadas em números muito diferentes.
- A ideia é **escolher M e a função de hashing** de modo a diminuir o número de **colisões**;
  - ou seja, espalhar bem as chaves pelo intervalo 0 . . M−1.

Função de hashing modular

- Que funções de hashing são usadas na prática?
  - se as chaves são inteiros positivos, pode-se usar a **função modular** (resto da divisão por M):

Função de hashing modular

- Que funções de hashing são usadas na prática?
  - se as chaves são inteiros positivos, pode-se usar a **função modular** (resto da divisão por M):

```
int hash(int key) {
    return key % M;
}
```

Função de hashing modular

- Que funções de hashing são usadas na prática?
  - se as chaves são inteiros positivos, pode-se usar a **função modular** (resto da divisão por M):

```
int hash(int key) {
    return key % M;
}
```

- Exemplos com M = 100 e com M = 97

key	hash (M = 100)	hash (M = 97)
212	12	18
618	18	36
302	2	11
940	40	67
702	2	23
704	4	25
612	12	30
606	6	24
772	72	93
510	10	25
423	23	35
650	50	68
317	17	26
907	7	34
507	7	22
304	4	13
714	14	35
857	57	81
801	1	25
900	0	27
413	13	25
701	1	22
418	18	30
601	1	19

Modular hashing

Função de hashing modular

- Em hashing modular, é bom que M seja **primo**
  - por algum motivo não óbvio
- No caso de strings, pode-se iterar hashing modular sobre os caracteres da string

Função de hashing modular

```
int h = 0;
for (int i = 0; i < s.length(); i++)
    h = (31 * h + s.charAt(i)) % M;
```

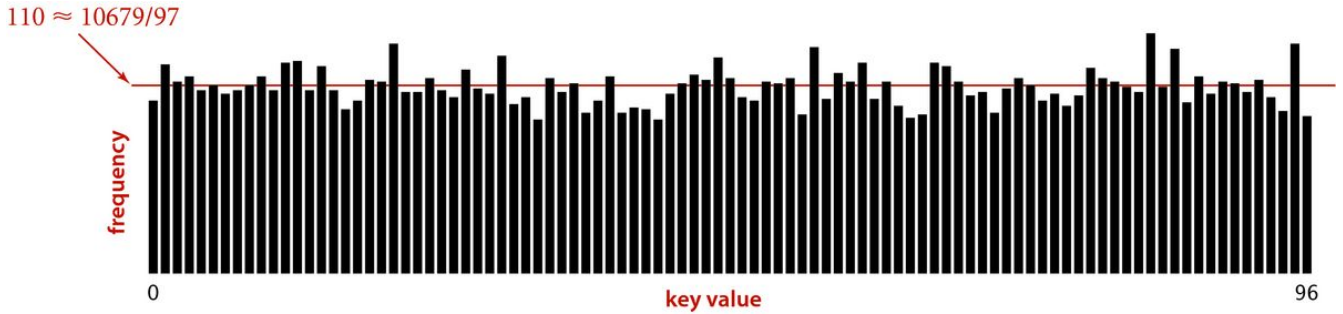
- No lugar do multiplicador 31, poderia usar qualquer outro inteiro R, de preferência primo, mas suficientemente pequeno para que os cálculos não produzam **overflow**.

Função de hashing ideal

- Que a função de hashing possa ser calculada eficientemente e espalhe bem as chaves pelo intervalo 0 . . M−1.
  - Exemplo: valores hash calculados a partir do hashCode() padrão do Java para um conjunto de palavras (excluídas as repetidas), com M = 97.

Função de hashing ideal

- No histograma, cada barra dá o número de palavras que têm o valor hash indicado na ordenada. O histograma sugere que a função espalha bem as palavras.



Hash value frequencies for words in *Tale of Two Cities* (10,679 keys, $M = 97$)

Hipótese do Hashing Uniforme

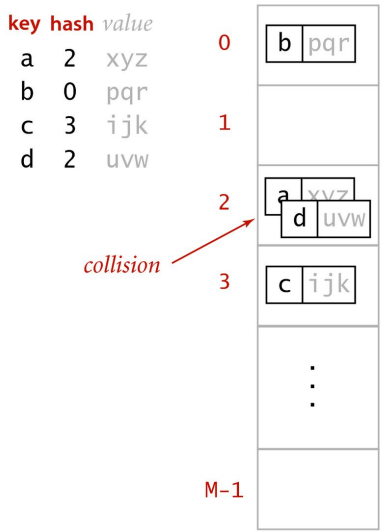
- Supõe-se que as funções de hashing distribuem as chaves pelo intervalo de inteiros 0 . . M−1 de maneira uniforme (todos os valores hash são igualmente prováveis) e independente.
- Na verdade, **nenhuma função determinística satisfaz a Hipótese do Hashing Uniforme**.

Hipótese do Hashing Uniforme

- No entanto, a hipótese é útil porque permite fazer cálculos para prever o **desempenho aproximado** de tabelas de hash.
- A função de hashing que do exemplo anterior **parece ser aproximadamente uniforme**.

Tratamento de Colisões

- Agora que cuidamos das funções de hashing, podemos tratar de **métodos de resolução de colisões**.
- É preciso ter meios de resolver colisões.



Hashing: the crux of the problem

Hashing com sondagem linear

- Um jeito **simples** de resolver colisões é com a sondagem linear (linear probing):
  - se uma posição da tabela estiver ocupada, tente a próxima!
  - precisa ter N ≤ M e portanto $\alpha \leq 1$.

Hashing com sondagem linear

- Exemplo: Dada a tabela de hashing de tamanho m = 6, e entradas limitadas a 3 por chave, incluir os elementos {36, 18, 31, 24, 30, 37, 49, 15, 17, 20}, conforme a sequência de entrada (utilizando a função de hashing pelo resto da divisão inteira)

Chave	Elementos				
0	36	18	24	~~30~~	~~15~~
1	31	30	37	~~49~~	
2	49	15	17	~~20~~	

Hashing com sondagem linear

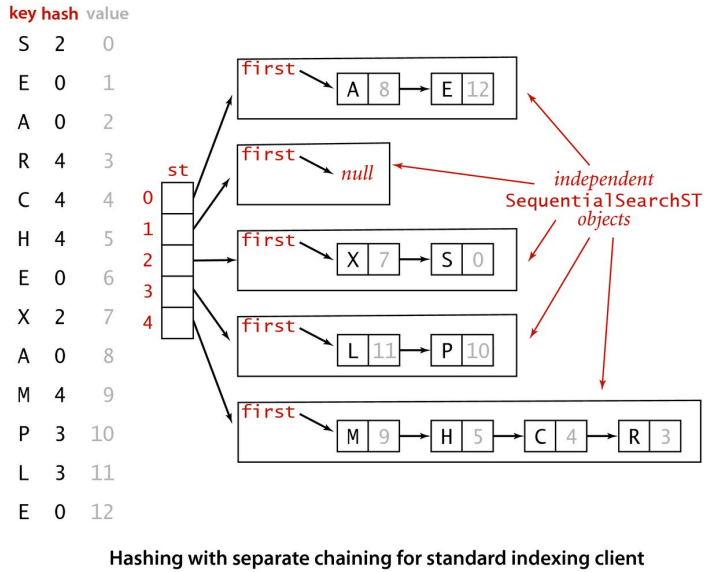
- Exemplo: Dada a tabela de hashing de tamanho m = 6, e entradas limitadas a 3 por chave, incluir os elementos {36, 18, 31, 24, 30, 37, 49, 15, 17, 20}, conforme a sequência de entrada (utilizando a função de hashing pelo resto da divisão inteira)

Chave	Elementos				
0	36	18	24	~~30~~	~~15~~
1	31	30	37	~~49~~	
2	49	15	17	~~20~~	

Redimensionar!!

Hashing com encadeamento

- Uma **solução elegante** é a resolução de colisões por **encadeamento** (separate chaining):
  - **M listas ligadas**, cada uma implementa uma tabela de símbolos.
  - Em geral, N > M e portanto $\alpha > 1$.



Hashing with separate chaining for standard indexing client

Referência

- R. Sedgewick, **Algorithms in C** (parts 1-4), 3rd. edition, Addison-Wesley/Longman, 1998.
  - sec.3.4, p.458 (Resumo). Código fonte, documentação, e dados de teste de todos os programas do livro: veja algs4.cs.princeton.edu/code/.

Resumo

- Conceitos preliminares
- Tabela de espalhamento (hashing)
- Funções de espalhamento (hashing)
- Tratamento de colisões

Tabela Hash (hashing)

Manassés Ribeiro
manasses.ribeiro@ifc.edu.br